IMPLEMENTASI ALGORITME TRIVIUM STREAM CIPHER DAN SHAMIR'S SECRET SHARING PADA *FILE PORTABLE DOCUMENT FORMAT (PDF)*

SKRIPSI

Disusun oleh:
Muhammad Wahyu Rizqi Pratama
NIM: 145150201111062

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
TAHUN 2018

PENGESAHAN

IMPLEMENTASI ALGORITME TRIVIUM STREAM CIPHER DAN SHAMIR'S SECRET SHARING PADA *FILE PORTABLE DOCUMENT FORMAT (PDF)*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh :
Muhammad Wahyu Rizqi Pratama
NIM: 145150201111062

Skripsi ini telah diuji dan dinyatakan lulus pada
2 Agustus 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Ari Kusyanti, S.T, M.Sc	Mahendra Data, S. Kom., M.Kom
NIP : 19831228 201803 2 002	NIK : 201503 861117 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP : 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang,

Muhammad Wahyu Rizqi Pratama
NIM: 145150201111062

KATA PENGANTAR

Assalamualaikum Wr. Wb. Alhamdulillah, puji syukur penulis panjatkan kehadirat Allah SWT atas limpahan rahmat dan hidayahnya sehingga skripsi ini dapat terselesaikan dengan baik. Skripsi ini disusun sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya.

Penulis menyadari bahwa tanpa bantuan, bimbingan, serta dorongan dari semua pihak, penyelesaian skripsi ini tidak mungkin bisa terwujud. Pada kesempatan ini penulis menyampaikan rasa terima kasih sebesar-besarnya kepada:

1. Ibu Anis Mahmudah dan Bapak Abdul Rachman yang memberikan dukungan dan doa setiap saat.
2. Bapak Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D selaku Dekan Fakultas Ilmu Komputer Universtitas Brawijaya.
3. Bapak Tri Astoto Kurniawan , S.T, M.T, Ph.D selaku Ketua Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya.
4. Bapak Agus Wahyu Widodo , S.T, M.Cs selaku Ketua Program Studi Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya .
5. Ibu Ari Kusyanti, S.T, M.Sc selaku dosen pembimbing pertama yang telah memberikan bimbingan, saran, motivasi, dan pengarahan dalam penyusunan skripsi ini.
6. Bapak Mahendra Data, S.Kom., M.Kom selaku dosen pembimbing kedua yang telah memberikan bimbingan, saran, motivasi, dan pengarahan dalam penyusunan skripsi ini.
7. Bapak, Ibu dosen serta segenap staff dan karyawan Jurusan Teknik Informatika yang telah membantu dalam menyelesaikan skripsi ini.
8. Semua pihak yang telah memberikan bantuan baik secara langsung maupun tidak langsung dalam penyusunan skripsi ini.

Dalam penyusunan skripsi ini penulis menyadari bahwa skripsi ini belum sempurna karena keterbatasan ilmu dan kendala kendala lain yang terjadi selama pengerjaan skripsi ini. Semoga tulisan ini dapat bermanfaat dan dapat digunakan untuk pengembangan lebih lanjut.

Wassalamualaikum Wr. Wb.

Malang,

Penulis

wahyurizqi363@gmail.com

ABSTRAK

Mengamankan data dalam bentuk *file* dengan mengenkripsi *file* tersebut merupakan salah satu cara untuk mengamankan data dalam konsep kriptografi, sehingga orang lain yang tidak berhak untuk mengetahui *file* yang sifatnya pribadi atau rahasia. Dalam penelitian ini mengimplementasikan keamanan data menggunakan dua algoritme kriptografi yakni Trivium Stream Cipher dan Shamir's Secret Sharing. Kedua algoritme tersebut diterapkan pada *file* dokumen berformat **.pdf*. *File* dokumen berformat **.pdf* merupakan format *file* yang banyak digunakan saat pengiriman *file* dikarenakan dapat didistribusikan pada komputer manapun dengan tampilan yang sama. Namun dengan semakin berkembangnya teknologi, keamanan pada file ini juga semakin melemah seperti penyadapan ataupun kehilangan. Maka diperlukan metode yang mampu meningkatkan keamanan *file* dokumen berformat **.pdf* ini. Diharapkan dengan menggabungkan dua algoritme ini dapat meningkatkan keamanan pada *file* berformat **.pdf*. Dari pengujian yang telah dilakukan sistem ini membutuhkan waktu eksekusi rata-rata 2,363 detik untuk *file* ukuran 15 Kb, 6,384 detik untuk *file* ukuran 31 Kb dan 12,456 detik untuk *file* ukuran 40 Kb.

Kata kunci: *kriptografi,* Shamir's Secret Sharing*,* Trivium Stream Cipher*, confidentiality dan availability.*

ABSTRACT

*Securing data in the form of files by encrypting the files is one way to secure data in the concept of cryptography, so that other people who are not entitled to find files that are private or confidential. In this study implementing data security using two cryptographic algorithms namely Trivium Stream Cipher and Shamir's Secret Sharing. Both algorithms are applied to a *.pdf format document file. * .pdf document file format is a file format that is widely used when sending files because it can be distributed on any computer with the same look. But with the development of technology, the security of this file also weakens like tapping or losing. Therefore we need a method that can improve the security of this * .pdf format document file. It is expected that by combining these two algorithms can improve security in * .pdf format files. From the testing that has been carried out this system requires an average execution time of 2.363 seconds for a file size of 15 Kb, 6.384 seconds for a file size of 31 Kb and 12.456 seconds for a file of size 40 Kb.*

Keywords: *cryptography, Shamir's Secret Sharing, Trivium Stream Cipher, confidentiality and availability.*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar belakang

Pada saat ini banyak sekali media yang memudahkan kita untuk mengirim *file* dokumen sehingga kita tidak perlu untuk langsung datang ke tempat tujuan. Sehingga dapat menghemat waktu, biaya dan tenaga. Salah satu *file* dokumen yang banyak digunakan adalah dalam format **.pdf* terlebih dalam suatu bisnis maupun bidang yang lainya seperti akademik, organisasi, kenegaraan, dan informasi keamanan. *File* dokumen tersebut dikirimkan atau disimpan ke berbagai media transfer dan *sharing*, seperti halnya *file sharing*, aplikasi pesan, email, dan *usb*. Namun dengan semakin berkembangnya dunia teknologi, maka semakin meningkat pula ancaman terhadap *file* dokumen tersebut seperti halnya media-media yang digunakan seringkali dapat disadap oleh pihak lain. Maka dibutuhkan pula cara untuk meningkatkan keamanan pada *file* dokumen tersebut. *File* yang dikirimkan atau disimpan melalui media *online* maupun *offline* mengandung sebuah informasi yang berguna bahkan sangat penting dan rahasia bagi penerimanya. Informasi yang sangat penting akan menjadi berbahaya apabila orang yang tidak mempunyai hak mengetahui isi dari informasi tersebut. Keamanan informasi menjadi hal penting agar isi informasi yang di kandung tidak diketahui oleh sembarang orang.

Kriptografi adalah ilmu yang mempelajari bagaimana menyembunyikan isi sebuah data. Penyembunyian isi sebuah berkas ini bertujuan untuk menjaga kerahasiaan data maupun isi data tersebut dari orang-orang yang tidak berhak dalam membaca pesan (Hamdani, 2012). Kriptografi memiliki berbagai konsep untuk melindungi isi data. Konsep-konsep tersebut dapat di padukan sehingga menghasilkan keamanan yang lebih tinggi. Salah satu konsep kriptografi yang dapat digunakan untuk menjaga keamanan data adalah Trivium Stream Cipher, konsep ini merupakan *synchronous stream cipher* yang dipelopori oleh Christophe De Canniere dan Bart Peneel, dan diikutsertakan pada kompetisi *eSTREAM*, dan telah dipilih sebagai bagian dari portofolio untuk *cipher* yang berorientasi pada perangkat keras. *Cipher* ini membangkitkan sampai dengan 2^{64} bit keluaran dari kunci dengan panjang 80 bit dan *IV* dengan panjang 80 bit. Trivium memiliki *state* internal sebesar 288 bit. *State* ini terdiri dari tiga *shift register* dengan panjang yang berbeda-beda. Pada setiap *round*, satu bit digeser ke masing-masing dari tiga *shift register* menggunakan kombinasi *non-linear* dari *tap* dan satu *register* lain. Satu bit keluaran dihasilkan untuk menginialisasi *cipher* ini, kunci dan *initial value* ditulis ke dua dari tiga *shift register* yang ada, dengan bit yang tersisa dimulai pada pola yang tetap, kemudian *state* dari *cipher* diperbaharui 4x288 = 1152 kali, sehingga setiap bit dari *state* internal bergantung pada setiap bit dari kunci dan *IV* secara tidak linear dan komplek. Metode ini dapat menambah keamanan data, dikarenakan belum ada kriptanalisis yang efektif untuk memecahkanya (Christophe De Canni`ere, 2007).

Konsep kriptografi yang lain adalah *secret sharing scheme*. S*ecret sharing scheme* merupakan suatu metode untuk melakukan *split* informasi tersebut

kedalam beberapa bagian yang disebut bagian *(shares)*, untuk di bagikan kepada beberapa penerima *(participants),* dengan suatu aturan tertentu. *Secret sharing* juga menangani masalah pendistribusian kunci rahasia yang telah di bagi dengan mengizinkan t (jumlah bagian yang di perlukan agar pesan dapat dibaca) dari n (jumlah bagian dari pesan) pengguna di mana t ≤ n untuk melakukan kunci awal. Skema *secret sharing* diperkenalkan oleh Blakley dan Shamir sebagai solusi untuk mengamankan kunci kriptografi. *Secret sharing* dapat juga digunakan untuk situasi apapun dimana akses kepada informasi harus terbatas, atau harus memiliki izin terlebih dahulu. Salah satu metode s*ecret sharing scheme* adalah Shamir's Secret Sharing. Metode ini dapat menambah keamanan informasi maupun kunci enkripsi dan deskripsi, hal ini dikarenakan informasi utama maupun kunci rahasia utama hanya bisa didapatkan jika dilakukan rekonstruksi dengan jumlah yang sudah di tentukan (Shamir, 1979).

Pada penelitian sebelumnya dilakukan oleh Dimas Aulia Trianggana tentang Implementasi algoritme *Blowfish* dan algoritme *Twofish* Pada proses enkripsi dan dekripsi, objek dari penelitian ini adalah *file* dokumen dengan format **.doc, *.xls, *.ppt* dan menghasilkan perbandingan waktu antara kedua algoritme tersebut, yakni dibuktikan algoritme *Blowfish* lebih cepat dibandingkan algoritme *Twofish*. (Dimas Aulia Trianggana, 2015). Dari Penelitian di atas masih menggunakan algoritme yang sudah lama dan sudah ditemukan kriptanalisis yang mampu memecahkan kedua algoritme tersebut. Penulis mengusulkan untuk mengganti algoritme tersebut menggunakan algoritme Trivium Stream Cipher sebagai standar baru yang dilakukan oleh *NIST* dalam proyek eSTREAM dan ditambahkan algoritme Shamir's Secret Sharing pada objek baru yakni *file* dokumen berformat **.pdf*. *NIST* adalah sebuah organisasi yang mengatur tentang standarisasi sebuah kemanan data di internet, di mana satandar sebuah data akan diperbarui setiap waktu dan diterapkan di seluruh dunia. Diharapkan dengan mengimplemetasikan algoritme Trivium Stream Cipher *dan* Shamir's Secret Sharing dapat dipadukan sehingga menghasilkan keamanan yang lebih tinggi dan sulit untuk ditembus khususnya didalam hal enkripsi *file* dokumen berformat **.pdf*, oleh karena itu, penelitian ini mengangkat tema "Implementasi Menggunakan Algoritme Trivium Stream Cipher dan Shamir's Secret Sharing pada *File Document Format (PDF)*".

1.2 Rumusan masalah

Berdasarkan latar belakang yang telah dikemukakan maka permasalahan yang akan dibahas dalam penelitian ini adalah :

1. Bagaimanakah implementasi algoritme Trivium Stream Cipher *dan* Shamir's Secret Sharing pada *file* dokumen berformat **.pdf* ?
2. Bagaimanakah validasi hasil pemrosesan *keystream,* enkripsi dan deskripsi pada algoritme Trivium Stream Cipher?
3. Bagaimanakah validasi hasil pemrosesan *split* dan *reconstruct* pada algoritme Shamir's Secret Sharing?

4. Bagaimanakah validasi hasil *file* dokumen berformat **.pdf* yang sudah diterapkan algoritme enkripsi dan *secret sharing*?
5. Berapa waktu yang dibutuhkan untuk proses algoritme Trivium Stream Cipher dan Shamir's Secret Sharing pada *file* dokumen berformat **.pdf*?

1.3 Tujuan

Adapun tujuan dari penelitian ini adalah sebagai berikut :

1. Melakukan penerapan algoritme Trivium Stream Cipher *dan* Shamir's Secret Sharing pada *file* dokumen berformat **.pdf*.
2. Melakukan validasi terhadap hasil pemrosesan *keystream,* enkripsi dan deskripsi pada algoritme Trivium Stream Cipher.
3. Melakukan validasi terhadap hasil pemrosesan *split* dan *reconstruct* pada algoritme Shamir's Secret Sharing.
4. Melakukan validasi terhadap hasil *file* yang sudah diterapkan algoritme Trivium Stream Cipher *dan* Shamir's Secret Sharing.
5. Melakukan pengujian dan analisis terhadap waktu yang dibutuhkan untuk proses algoritme Trivium Stream Cipher *dan* Shamir's Secret Sharing pada *file* dokumen berformat **.pdf.*

1.4 Manfaat

Adapun manfaat yang diharapkan adalah dapat memberikan alternatif keamanan *file* menggunakan kriptogafi dengan algoritme Trivium Stream Cipher *dan* Shamir's Secret Sharing yang dapat diterapkan pada *file* dokumen berformat **.pdf,* sehingga dapat menambah aspek *confidentiality* dan *availability.*

1.5 Batasan masalah

Adapun batasan masalah dalam penulisan skripsi ini adalah :

1. Algoritme membagi informasi atau kunci rahasia yang digunakan, yaitu metode Shamir's Secret Sharing dengan jumlah *split* 3 dan jumlah *reconstruct* 2.
2. *File* dokumen yang digunakan dalam penelitian ini adalah *file* dokumen berformat **.pdf* dengan ketentuan jumlah halaman tidak lebih dari 1 halaman, dan ukuran *file* tidak lebih dari 40 kb.
3. *Library* yang digunakan untuk membaca dan menulis *PDF* adalah *i-Text*.
4. Sistem yang dibuat hanya untuk proses enkripsi dan *secret sharing* dalam satu laptop.

1.6 Sistematika pembahasan

Bab I : Pendahuluan

Dalam bab ini penulis menguraikan latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat dan sistematika pembahasan dari penelitian

"Implementasi Algoritme Trivium Stream Cipher dan Shamir's Secret Sharing Pada *File Portable Document Format (PDF)*".

Bab II : Kajian Pustaka dan Dasar Teori

Dalam bab ini penulis membahas mengenai kajian pustaka dan teori-teori yang diperlukan didalam penelitian "Implementasi Algoritme Trivium Stream Cipher dan Shamir's Secret Sharing Pada *File Portable Document Format (PDF)*".

Bab III : Metodologi Penelitian

Dalam bab ini penulis membahas metode atau langkah-langkah yang dilakukan dalam penelitian "Implementasi Algoritme Trivium Stream Cipher dan Shamir's Secret Sharing Pada *File Portable Document Format (PDF)*".

Bab IV : Perancangan

Dalam bab ini penulis membahas *design* program yang disajikan dalam bentuk diagram alir dan juga design struktur data yang akan digunakan dalam program penelitian "Implementasi Algoritme Trivium Stream Cipher dan Shamir's Secret Sharing Pada *File Portable Document Format (PDF)*".

Bab V : Implementasi

Dalam bab ini penulis membahas hasil *design* ke dalam bentuk *source code* program antara lain *procedure* dan *function* yang dipakai, serta analisa hasil uji coba dari penelitian "Implementasi Algoritme Trivium Stream Cipher dan Shamir's Secret Sharing Pada *File Portable Document Format (PDF)*".

Bab VI : Pengujian dan Pembahasan

Bab ini akan menjelaskan skenario pengujian, hasil pengujian beserta analisa dari penelitian "Implementasi Algoritme Trivium Stream Cipher dan Shamir's Secret Sharing Pada *File Portable Document Format (PDF)*".

Bab VII : Penutup

Bab ini akan berisi kesimpulan yang merupakan rangkuman proses dan hasil dari keseluruhan penelitian dan saran yang berguna untuk pengembangan dan perbaikan untuk sistem yang ada.

BAB 2 LANDASAN KEPUSTAKAAN

Landasan kepustakaan berisi tentang uraian dan pembahasan dasar teori terkait dengan penelitian. Penulis mengkaji beberapa jurnal untuk dijadikan referensi, diantaranya adalah jurnal yang berjudul "*A new Challenge of hiding any encrypted secret message inside any Text/ASCII file or in MS word file: RJDA Algorithm*", oleh Rishav Ray, Jeeyan Sanyal, Debanjan Das, Asoke Nath. " *A Secure Authentication Protocol for RFID Based on Trivium*", oleh Jiezhong Gong, Gongliang Chen, Linsen Li, Jianhua Li. "*Sharing Secret Images Using Blakley's Concept*", oleh Ching-Nung Yang, Chih-Cheng Wu, Chih-Wei Chou.

2.1 Kajian Pustaka

Pada sub bab ini menjelaskan megenai perbedaan penelitian sebelumnya yang digunakan sebagai kajian pustaka dengan penelitian yang akan diteliti. Penulis menggunakan 3 macam penelitian yang sudah dilakukan sebelumnya sebagai rujukan. Berikut perbedaan dan persamaan penelitian sebelumnya dapat dilihat pada Tabel 2.1.

Tabel 2.1 Kajian Pustaka

No	Nama Peneliti, Tahun, dan Judul	Persamaan	Perbedaan	
			Penelitian sebelumnya	Penelitian penulis
1	Dimas Aulia Trianggana, Herlina Latipa Sari,(2015)," ANALISIS PERBANDINGAN KINERJA ALGORITME BLOWFISH DAN ALGORITME TWOFISH PADA PROSES ENKRIPSI DAN DEKRIPSI"	Menggunakan *objek file text*	Implementasi pada file Microsoft Word, menggunakan algoritme Blowfish dan Twofish	Implementasi pada *file Portable Document Format* (PDF), menggunakan algoritme Trivium *stream cipher*
2	Jiezhong Gong, Gongliang Chen, Linsen Li, Jianhua Li, (2011)," A Secure Authentication Protocol for RFID Based on Trivium"	Menggunakan algoritme *Trivium stream cipher*	Implementasi pada *protocol* RFID	Implementasi pada *file Portable Document Format* (PDF)

3	Ching-Nung Yang, Chih-Cheng Wu, Chih-Wei Chou, (2013), "Sharing Secret Images Using Blakley's Concept"	Menggunakan *secret sharing scheme*	Implementasi pada gambar, menggunakan algoritme *blakley secret sharing*	Implementasi pada file *portable document format* (PDF), menggunakan algoritme Shamir's Secret Sharing

2.2 Dasar Teori

2.2.1 Pengertian Kriptografi

Kriptografi (Cryptography) berasal dari bahasa Yunani, terdiri dari dua suku kata yaitu *kripto* dan *graphia*. *Kripto* sendiri mempunyai arti menyembunyikan, sedangkan graphia artinya tulisan. Kriptografi adalah ilmu yang mempelajari teknik-teknik matematika yang berhubungan dengan aspek keamanan informasi, seperti kerahasiaan data, keabsahan data, serta autentikasi data. Kriptografi dapat juga diartikan sebagai metode keamanan untuk melindungi informasi dengan menggunakan kata-kata sandi yang hanya bisa dimengerti oleh orang yang berhak mengakses informasi tersebut. Kriptografi digunakan untuk melindungi informasi yang melalui jaringan komunikasi, seperti *landline* (kabel dibawah tanah), satelit komunikasi, dan fasilitas *microwave* (gelombang mikro). Prosedur-prosedur kriptografi dapat digunakan untuk *authentifikasi* pesan, *digital signature*, dan indentifikasi pribadi (Amin, 2016).

Dalam bidang ilmu kriptografi, dikenal dua istilah yaitu enkripsi dan dekripsi. Enkripsi merupakan proses pengacakan naskah asli *(plaintext)* menjadi naskah acak yang sulit dibaca *(ciphertext)* sedangkan dekripsi adalah proses pengembalian cipher ke dalam plain. Kedua proses ini membutuhkan satu komponen yang disebut kunci. Kunci dan plain akan dikombinasikan dengan serangkaian rumus matematika dalam algoritme enkripsi sehingga menghasilkan *cipher*. Kunci yang digunakan untuk proses dekripsi bisa merupakan kunci yang sama saat proses enkripsi (kunci simetris) maupun kunci yang berbeda saat proses enkripsi (kunci asimetris) (Ariyus, 2008). Berdasarkan cara melakukan enkripsi, algoritme enkripsi dibagi menjadi algoritme yang melakukan enkripsi dengan bit-bit data *(stream cipher)* dan algoritme yang melakukan enkripsi dengan blok-blok data *(block cipher)* (Kurniawan, 2004). Proses enkripsi dan deskripsi dapat dilihat pada Gambar 2.1.

Gambar 2.1 Proses enkripsi dan dekripsi

2.2.2 Tujuan Kriptografi

Tujuan kriptografi adalah melindungi data dari ancaman yang semakin bertambah dikarenakan semakin meluasnya akses melalui internet . Aspek-aspek keamanan data dalam kriptografi adalah sebagai berikut (Ariyus, 2008).

1. *Confidentiality*

   Merupakan usaha untuk menjaga kerahasiaan data dari pihak-pihak yang tidak mempunyai hak atas data tersebut. Aspek keamanan data menjadi sensitif dalam *e-commerce* dan militer. Serangan dalam aspek ini antara lain dilakukan dengan penyadapan, misalnya *sniffer* atau *logger*.

2. *Integrity*

   Merupakan usaha untuk memastikan bahawa informasi yang dikirim tidak mengalami modifikasi oleh pihak yang tidak mempunyai hak. Serangan dapat berupa pengubahan data yang dilakukan dengan *spoofing*.

3. *Availability*

   Merupakan suatu usaha untuk menjamin ketersediaan informasi ketika dibutuhkan

4. *Authentication*

   Merupakan proses validasi yang hanya mengijinkan pihak tertentu untuk dapat mengakses informasi. *Authentication* ini bisa diterapkan dengan menggunakan *password* atau teknik *digital signature*.

5. *Acces Control*

   Merupakan proses untuk mengatur masalah akses informasi. Hal ini biasanya berhubungan dengan masalah autentikasi dan privasi.

6. *Privacy*

   Merupakan aspek yang mementingkan hal-hal yang bersifat pribadi.

7. *Authority*

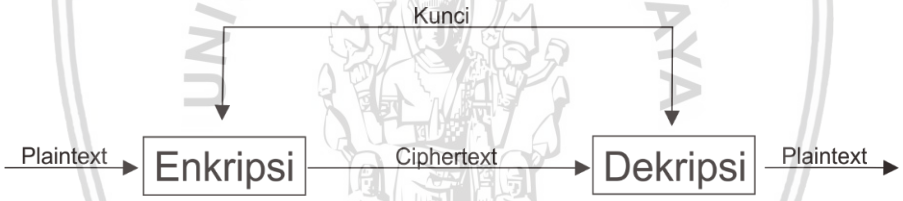
   Merupakan proses untuk mengatur hak akses dari pihak-pihak yang dapat mengakses informasi.

Dengan beberapa aspek keamanan dalam kriptografi tersebut menjadikan kriptografi dibagi menjadi 2 kelompok menurut algoritmenya, yaitu simetris dan asimetris. Algoritme simetris merupakan algoritme yang mempunyai kunci yang sama untuk proses enkripsi dan dekripsi. Sedangkan algoritme asimetris merupakan algoritme yang mempunyai kunci yang berbeda untuk proses enkripsi dan dekripsi.

2.2.3 Kriptografi Simetris

Pada sistem kriptografi simetris, kunci yang digunakan untuk proses enkripsi dan dekripsi adalah sama. Keamanan sistem kriptografi simetris terletak pada kerahasiaan kunci. Nama lain untuk kriptografi simetri adalah kriptografi kunci privat *(private key cryptography)* atau kriptografi konvensional *(convensional cryptography)*. Contoh dari kriptografi simetris adalah *DES, AES,IDEA, GRAIN, TRIVIUM, A5, RC4* dan lain-lain. Algoritme kriptografi simetris dapat dikelompokkan menjadi dua kategori, yakni s*tream cipher* dan *block cipher.* Pada algoritme *stream cipher* dalam proses penyandiannya berorientasi pada satu bit atau satu *byte* data. Sedang pada algoritme *blok cipher*, proses penyandianya berorientasi pada sekumpulan bit atau *byte* data (per-blok).

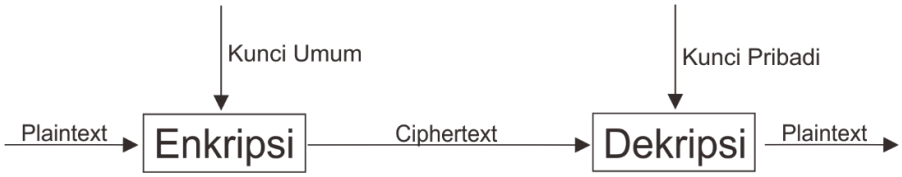
Kelebihan algoritme simetris ini adalah di dalam kecepatanya dalam proses enkripsi dan dekripsinya, dibandingkan algoritme asimetris. Sedangkan kelemahan algoritme ini adalah permasalahan distribusi kunci (*key distribution*) yakni proses enkripsi dan dekripsi menggunakan kunci yang sama. Sehingga muncul persoalan apabila saat pengiriman kunci pada media yang tidak aman seperti internet dan kunci tersebut hilang atau sudah diketahui oleh orang yang tidak berhak, maka kriptosistem ini sudah tidak aman lagi (Tarbudi, 2013). Proses kriptografi simetri dapat dilihat pada Gambar 2.2.



Gambar 2.2 Kriptografi Simetri

2.2.4 Kriptografi Asimetris

Pada sistem kriptografi asimetris, kunci untuk proses enkripsi tidak sama dengan kunci untuk proses dekripsi. Nama lain untuk kriptografi asimetris adalah kriptografi kunci publik *(public key cryptography)*, sebab kunci untuk enkripsi tidak rahasia dan dapat diketahui oleh siapapun, sementara kunci untuk dekripsi hanya diketahui oleh penerima pesan. Contoh dari kriptografi simetris adalah *RSA, ElGamal, McEliece, LUC, DSA*. Keuntungan utama dari algoritme ini adalah memberikan jaminan keamanan kepada siapa saja yang melakukan pertukaran informasi meskipun diantara mereka tidak ada kesepakatan mengenai kemanan data terlebih dahulu maupun saling tidak mengenal satu sama lain (Tarbudi, 2013). Proses kriptografi asimetri dapat dilihat pada Gambar 2.3.



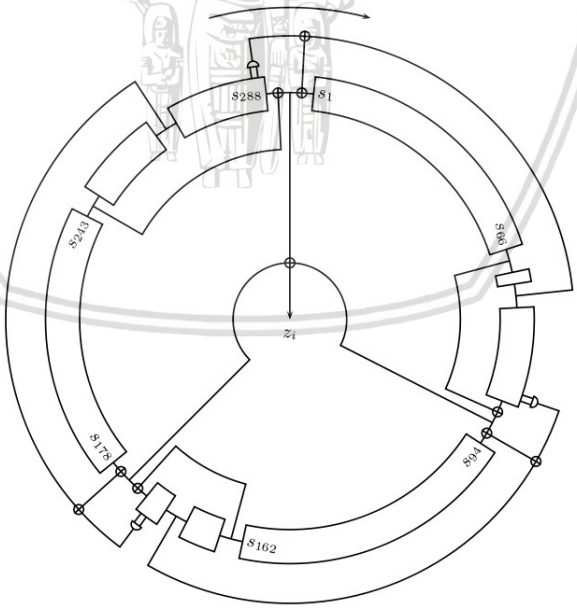
Gambar 2.3 Kriptografi Asimetri

2.2.5 Trivium Stream Cipher

Trivium merupakan *synchronus stream cipher* yang didesain untuk dapat membangkitkan kunci *stream* hingga 2^{64} bit dari 80 bit kunci rahasia dan 80 bit *initial value (IV)*. Seperti kebanyakan *stream* kebanyakan *stream cipher* lainya, proses pembangkitan ini mengandung dua fase :

- Fase *inialisasi state* internal menggunakan *IV* dan kunci rahasia
- Perubahan kondisi *state* internal untuk menghasilkan kunci *stream*

Stream cipher ini dibuat pada tahun 2005 oleh Christophe De Canni're dan Bart Preneel untuk diikutsertakan pada European project *eSTREAM*. Pada kontes tersebut Trivium berhasil melewati fase pertama dan dimasukkan dalam fokus khusus *hardware* portofolio. Pada fase kedua Trivium memiliki *state* internal yang berukuran 288 bit dan kunci dengan panjang 80 bit. Meskipun *stream cipher* ini memiliki desain yang sederhana namun hingga kini belum ditemukan metode kriptanalisis yang efektif untuk memecahkanya. Fase pada algoritme Trivium Stream Cipher dapat dilihat pada gambar 2.4.



Gambar 2.4 Diagram Umum *Trivium Stream Cipher*

Sumber : (Christophe De Canni`ere, 2007)

2.2.6 Key and *Setup*

Trivium ini diinialisasikan dengan memasukkan 80 bit *IV* ke dalam *state* internal yang berukuran 288 bit, dan mengisi sisa bit *state* internal dengan 0, kecuali untuk s286, s287, s288. Kemudian state dirotasikan sebanyak 4 kali, tanpa proses pembangkitan bit kunci *stream*. Rumus dapat dilihat pada algoritme (2.1).

$(S_1, S_2, \ldots, S_{93}) \leftarrow (K_1, \ldots, K_{80}, 0, \ldots, 0)$

$(S_{94}, S_{95}, \ldots, S_{177}) \leftarrow (IV_1, \ldots, IV_{80}, 0, \ldots, 0)$

$(S_{178}, S_{279}, \ldots, S_{288}) \leftarrow (0, \ldots, 0,1,1,1)$

For i = 1 to 4 . 288 do

$t1 \leftarrow S_{66} + S_{91} \cdot S_{92} + S_{93} + S171$

$t2 \leftarrow S_{162} + S_{175} \cdot S_{176} + S_{177} + S_{264}$

$t3 \leftarrow S_{243} + S_{286} \cdot S_{287} + S_{288} + S_{69}$

$(S_1, S_2, \ldots, S_{93}) \leftarrow (t_3, S_1, \ldots, S_{92})$

$(S_{94}, S_{95}, \ldots, S_{177}) \leftarrow (t_1, S_{94}, \ldots, S_{176})$

$(S_{178}, S_{279}, \ldots, S_{288}) \leftarrow (t_2, S_{178}, \ldots, S_{287})$

End for (2.1)

2.2.7 Key Stream Generation

Trivium memiliki 288 bit *state* internal yang di notasikan sebagai (s1,. . . . ,s288). Proses pembangkitan kunci *stream* mengandung proses perulangan yang mengekstrak nilai dari 15 bit spesifik *state* dan menggunakanya untuk meng*update* 3 bit dari *state* dan untuk menghitung 1 bit dari kunci zi. Bit-bit *state* kemudian dirotasikan dan proses berulang hingga $N \leq 2^{64}$ kunci *stream* yang dibangkitkan. Panjang *keystream* dapat disesuaikan dengan keinginan *user.* Rumus dapat dilihat pada algoritme (2.2).

for i = 1 to N do

$t_1 \leftarrow S_{66} + S_{93}$

$t_2 \leftarrow S_{162} + S_{177}$

$t_3 \leftarrow S_{243} + S_{288}$

$z_i \leftarrow t_1 + t_2 + t_3$

$t_1 \leftarrow t1 + S_{91} \cdot S_{92} + S_{171}$

$t_2 \leftarrow t_2 + S_{175} \cdot S_{176} + S_{264}$

$t_3 \leftarrow t_3 + S_{286} \cdot S_{287} + S_{69}$

$(S_1, S_2, \ldots, S_{93}) \leftarrow (t_3, S_1, \ldots, S_{92})$

$(S_{94}, S_{95}, \ldots, S_{177}) \leftarrow (t_1, S_{94}, \ldots, S_{176})$

$(S_{178}, S_{279}, \ldots, S_{288}) \leftarrow (t_2, S_{178}, \ldots, S_{287})$

End for (2.2)

2.2.8 Secret Sharing Scheme

Secret sharing scheme adalah konsep kriptografi yang digunakan untuk membagi rahasia ke beberapa bagian. Rahasia yang dimaksud dalam hal ini adalah sebuah bilangan bagian-bagian yang dihasilkan dapat digunakan untuk membangun bilangan awal. Bilangan tersebut merupakan representasi sebuah pesan rahasia yang telah dirubah ke bentuk bilangan atau nilai kunci rahasia. Konsep ini dapat meningkatkan keamanan dikarenakan jika hanya satu kunci saja yang didapat oleh penyadap atau orang yang tidak berhak, maka pesan tersebut tidak akan bisa terbaca sehingga keamanan pesan akan tetap terjaga. Pembentukan bilangan awal dapat dilakukan dengan beberapa metode. Salah satu metode yang dapat digunakan adalah Shamir's Secret Sharing (Satria Prayudi, 2015).

Shamir's Secret Sharing adalah metode pembagian skema rahasia ke dalam beberapa bagian dengan memecahkan permasalahan persamaan matematika. Biasanya metode ini menggunakan notasi (n,k). Nilai n menunjukkan jumlah bagian yang dihasilkan. Selanjutnya nilai k menunjukkan jumlah minimal bagian yang diperlukan untuk rekonstruksi bilangan awal. Biasanya metode ini menggunakan bilangan prima yang lebih besar dari bilangan rahasia dan setiap koefisien persamaan. Jumlah minimal k bilangan untuk rekonstruksi nilai rahasia S menggunakan persamaan matematika derajat k-1. Berikut adalah langkah-langkah metode Shamir's Secret Sharing untuk membagi bilangan rahasia S1, S2, ..., Sn ke beberapa bagian bilangan :

- Pilih bilangan prima (P), yang lebih besar dari nilai rahasia (S) dan lebih besar dari setiap koefisien persamaan untuk k partisipan. Komputasi-komputasi dihasilkan dalam modulus P sehingga perhitungan aritmatik dalam bidang terbatas.
- Pilih bilangan acak sebanyak k-1 dengan syarat angka lebih kecil dari bilangan prima P, misalkan a_1, a_2, ... , a_{k-1}, lalu nyatakan dalam polinomial. Rumus dapat dilihat pada persamaan (2.1)

$$S(x) \equiv M + a_i x^1 + a_2 x^2 + \cdots + a_{k-1} x^{k-1} (Mod\ P) \quad (2.1)$$

- Setiap partisipan ditentukan sebuah nilai x yang berbeda x1, x2, ... , xk-1 (mod P). Selanjutnya setiap partisipan memperoleh *share* masing-masing berdasarkan nilai xi yang dimasukkan ke persamaan S(xi). Rumus dapat dilihat pada persamaan (2.2)

$$y_i \equiv s(x_i) \quad (2.2)$$

Nilai xi dan yi yang didapat adalah bagian-bagian untuk n partisipan. Nilai rahasia S semula akan dihasilkan dari gabungan k nilai dari n partisipan. Beberapa cara dapat digunakan untuk menghasilkan nilai rahasia S dari persamaan diatas. Cara-cara yang digunakan adalah eliminasi *Gauss* dan *interpolasi lagrange*. Penelitian ini menggunakan *interpolasi lagrange* dalam menyelesaikan persamaan diatas.

2.2.9 Interpolation Lagrange

Polinomial lagrange digunakan untuk interpolasi *polinomial*. Untuk setiap pasangan xi dan yi yang berbeda, *polinomial lagrange* adalah *polinomial* derajat terendah yang mengasumsikan setiap nilai x, bersesuaian dengan nilai yi. Titik-titik tersebut dimasukkan ke dalam rumus yang disebarluaskan untuk membentuk persamaan *lagrange*. Rumus dapat dilihat pada persamaan (2.3) dan (2.4).

$$F(x) = \sum_{j=1}^{n} P_j\ (x) \tag{2.3}$$

Dimana,

$$F(x) = \ y_j \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k} \tag{2.4}$$

Kalkulasi diatas dalam modulus P, hal ini disebabkan agar kalkulasi dalam bidang terbatas. Nilai rahasia S dapat dihitung dengan memasukkan nilai x = 0, pada persamaan P(X). Nilai P(0) bersesuaian dengan nilai rahasia S, dikarenakan nilai tersebut merupakan koefisien x0 pada persamaan (2.5).

$$S(x) \equiv M + \ a_i x^1 + \ a_2 x^2 + \cdots + a_{k-1} x^{k-1} (Mod\ P) \tag{2.5}$$

2.2.10 PDF (Portable Document Format)

PDF adalah kependekan dari *Portable Document Format*, **.pdf* adalah format *file* yang dikembangkan oleh *Adobe sistem*. *PDF* pertama kali dibuat dimaksudkan untuk mempermudah pertukaran dokumen, keunggulan dari dokumen yang dibuat dengan format **.pdf* adalah tidak bisa dirubah secara langsung oleh penerima dokumen tidak seperti *file* dokumen dengan format **.doc, *.txt* atau yang lainnya. Namun dengan perkembangan teknologi yang semakin canggih, *file* dokumen yang dibuat dengan format **.pdf* bisa dirubah isinya dengan menggunakan bantuan *software PDF editor* yang sudah banyak beredar. Struktur umum *file PDF* terbagi menjadi 4 bagian utama, terdiri dari *header, body, cross-reference table*, dan *trailer* (Imad Fakhri Al Shaikhli, 2012).

PDF pertama kali dikenalkan pada tahun 1993, tingkat penggunaan dokument berformat PDF relatif rendah, dikarenakan pada saat itu aplikasi untuk membuat dokumen *PDF (Adobe Acrobat)* dan aplikasi untuk membacanya (*Acrobat Reader, sekarang Adobe Reader)* tersedia secara komersial , tidak di distribusikan secara gratis. Pada awalnya *PDF* tidak mendukung fasilitas pranalar luar, yang membuatnya kurang terintegrasi dengan w*orld wide web*, dan ukuranya yang semakin besar menjadikan lambat untuk diunduh (*download*) dengan tingkat kecepatan modem pada saat itu. Faktor lain yang menyebabkan kurang diminatinya *PDF* pada masa itu adalah format berkas *PDF* harus bersaing dalam tingkat penggunaanya dengan format lain seperti *Envoy, Common Ground Digital Paper.*

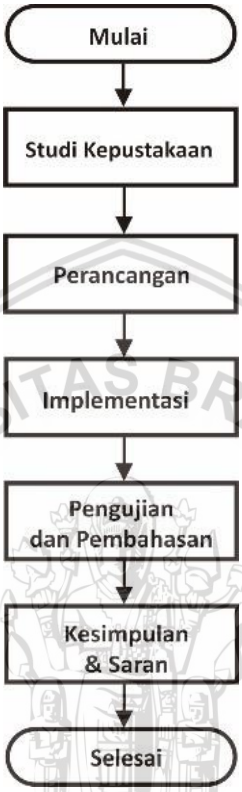
2.2.11 Java

Java adalah bahasa pemograman yang *sintaks* nya diturunkan dari bahasa pemograman *C* dan konsep objek *oriented*nya dipengarui dari bahasa pemograman *C++*. Konsep *objek oriented* pada *java* memungkinkan program yang dibangun memiliki cara kerja seperti cara manusia menggunakan bahasa sehari-hari. Karenanya, *java* termasuk bahasa pemograman yang terbilang sederhana dan mudah dipahami. Inovasi yang diciptakan dalam bahasa pemograman mengacu pada 2 hal mendasar, yaitu dapat beradaptasi pada lingkungan dan pemakaian yang sering berubah serta mampu memberikan perbaikan dan kemajuan dalam dunia pemograman (Schildt, 2007).

Java dibuat oleh James Ghosling, Patrick Naughton dan Mike Sheridan dari Sun Microsistem pada tahun 1991. Bahasa pemograman *java* kini terus berkembang dan menjadi salah satu bahasa pemograman yang populer dan sering digunakan, Selain itu *java* juga mampu dikembangkan melalui berbagai macam *paltform* seperti *Windows, Linux* maupun *Mac*.

BAB 3 METODOLOGI

Bab ini akan menjelaskan langkah-langkah dalam penelitian yang berjudul "Implementasi Algoritme Trivium Stream Cipher dan Shamir's Secret Sharing Pada *File Portable Document Format (PDF)*". Secara garis besar, tahapan penelitian dapat dilihat pada Gambar 3.1.



Gambar 3.1 Diagram Alir Penelitian

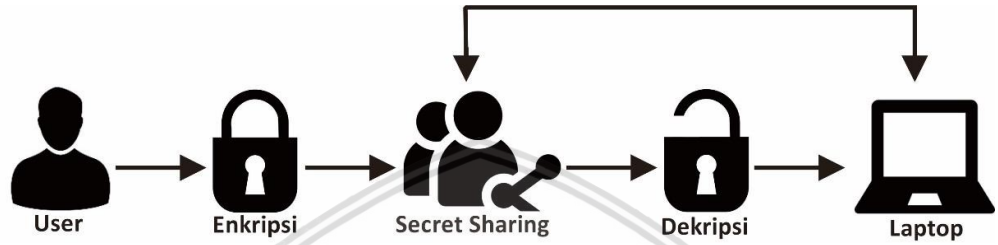
3.1 Studi Kepustakaan

Dalam pengerjaan penelitian, pemahaman mendalam akan konsep-konsep pada ilmu terkait akan sangat dibutuhkan. Teori utama yang perlu dipelajari lebih dalam adalah kriptografi dan *secret sharing*. Selain itu, pengetahuan akan penelitian sebelumnya yang membahas kriptografi dan *secret sharing* akan dikaji secara komprehensif. Hal ini bisa ditempuh dengan memperbanyak bacaan melalui jurnal, buku dan *website* resmi yang membahas topik seputar penelitian.

3.2 Perancangan Sistem

Penelitian ini bertujuan untuk membangun sistem keamanan dengan menerapkan algoritme enkripsi dan *secret sharing*. Tipe dari penelitian ini adalah implementatif. Tujuan dari tahap ini adalah untuk memberikan gambaran sistem yang akan dikembangkan. Tahap perancangan juga menjadi acuan yang digunakan ketika akan melakukan tahap implementasi. Pada tahap ini akan dijelaskan secara rinci kebutuhan-kebutuhan serta alur kerja pada setiap proses yang berjalan pada

sistem dalam sub-bab lingkungan sistem dan perancangan sistem. Secara umum, sistem skenario yakni dari *user* akan menginputkan *file* yang ingin diterapkan algoritme enkripsi dan *secret sharing* dan kemudian hasil dari enkripsi dan *secret sharing* akan disimpan dalam laptop, kemudian akan dilakukan proses rekonstruksi atau penggabungan *file* dengan cara mengambil *file* dari laptop dan kemudian akan diproses dengan algoritme *secret sharing* dan dekripsi. Proses terakhir adalah penyimpanan seluruh proses sistem pada laptop. Gambaran umum dari sistem dapat dilihat pada Gambar 3.2.



Gambar 3.2 Gambaran Umum Sistem

3.3 Implementasi

Perancangan sistem yang sudah dibuat akan di implementasikan pada tahap ini. Ada 3 macam implementasi yang dilakukan yaitu implementasi algortima Shamir's Secret Sharing, algoritme Trivium Stream Cipher, implementasi penyimpanan *file* di dalam komputer.

1. Implementasi algoritme Shamir's Secret Sharing
   Pada tahap implementasi ini, algoritme akan di tulis ke dalam kode program *java* yang nantinya digunakan untuk memecah *(split)* atau menggabungkan *(reconstruct) share* dari data *cipher* .
2. Implementasi algoritme Trivium Stream Cipher
   Pada tahap implementasi ini, algoritme akan ditulis ke dalam kode program *java* yang nantinya digunakan untuk enkripsi dan dekripsi data
3. Implementasi penyimpanan dan pengambilan *file* dalam laptop
   Pada tahap implementasi ini, algoritme akan ditulis ke dalam kode program *java* yang nantinya digunakan untuk menyimpan *file* hasil olahan dari algoritme Shamir's Secret Sharing dan Trivium Stream Cipher.

3.4 Pengujian dan Pembahasan

Tahap pengujian dan pembahasan akan berisi skenario pengujian, hasil pengujian dan analis yang didapat dari pengujian. Pengujian yang dilakukan adalah :

1. Pengujian algoritme Trivium Stream Cipher menggunakan 2 cara yakni pengujian *test vector* dan pengujian enkripsi atau dekripsi:
   - Pengujian pertama dengan cara melakukan pengujian *test vector*. Pengujian ini bertujuan untuk menyesuaikan nilai *keystream* yang dihasilkan sistem dengan *keystream* pada test vector yang dirancang oleh pembuat algoritme Trivium Stream Cipher. Pengujian ini dilakukan sebagai

validasi bahwa algoritme Trivium Stream Cipher yang diterapkan sudah sesuai dengan algoritme Trivium Stream Cipher yang dibuat oleh Christophe De Canni're dan Bart Preneel. Cara pengujian ini adalah dengan mencocokkan nilai *keystream* sistem dengan nilai *keystream test vector*.

- Pengujian kedua dengan cara enkripsi dan dekripsi. Pengujian ini bertujuan untuk mengetahui apakah algoritme yang menggunakan teknik enkripsi dan dekripsi sudah berjalan dengan baik. Pengujian enkripsi bisa dianggap berhasil jika data yang diamankan dapat berubah menjadi data acak yang sulit dibaca. Pengujian dekripsi bisa dianggap berhasil jika data acak dapat kembali menjadi data semula. Cara pengujian ini dilakukan dengan melihat hasil enkripsi dan menyesuaikan data sebelum proses enkripsi dengan data setelah proses dekripsi.

2. Pengujian algoritme Shamir's Secret Sharing dengan cara *split* dan *reconstruct*. Pengujian ini bertujuan untuk mengetahui apakah algoritme Shamir's Secret Sharing yang digunakan sudah berjalan dengan baik. Pengujian *split* bisa di anggap berhasil jika nilai data yang dipecah telah sesuai dengan nilai hitungan manual yang dilakukan oleh penulis. Pengujian *reconstruct* bisa dianggap berhasil jika bagian *(share)* yang dihasilkan, mampu untuk mengembalikan nilai data seperti semula. *Split* yang digunakan adalah 3 dan *reconstruct* yang digunakan adalah 2.
3. Pengujian kevalidan *file* yang ada di komputer. Pengujian ini bertujuan untuk mengetahui *file* yang dihasilkan oleh proses Shamir's Secret Sharing dan Trivium Stream Cipher dapat sesuai dengan *file* yang asli.
4. Pengujian waktu yang dibutuhkan pada proses enkripsi, dekripsi, *split* dan *reconstruct file PDF*. Pengujian ini bertujuan untuk mengetahui waktu proses keseluruhan sistem.

3.5 Kesimpulan & Saran

Pengambilan kesimpulan dan saran merupakan tahap terakhir pada penelitian ini. Kesimpulan diambil dari hasil pengujian terhadap aplikasi yang telah dibangun. Tahap terakhir penulisan adalah saran yang dimaksudkan untuk memperbaiki kekurangan yang terjadi dan menyempurnakan penulisan serta mengembangkan penelitian lebih lanjut.

BAB 4 PERANCANGAN

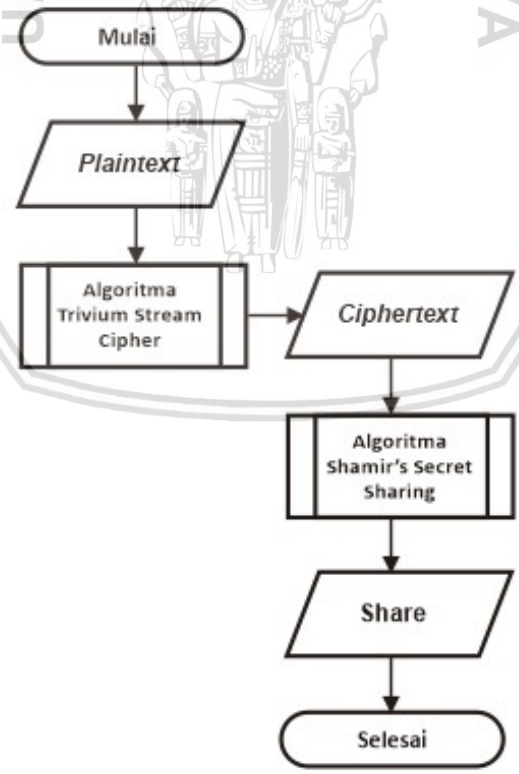
Bab empat menjelaskan tentang peranan perancangan sistem yang akan digunakan sebagai rujukan pada tahapan implementasi. Implementasi menjelaskan tentang perancangan sistem yang sudah di terapkan dalam kode program yakni mulai dari algoritme Trivium Stream Cipher sampai algoritme Shamir's Secret Sharing.

4.1 Perancangan

Perancangan sistem merupakan proses untuk melakukan tahap implementasi , dimana perancangan berperan terhadap kebutuhan sistem agar dapat diimplementasikan dengan benar.

4.1.1 Sistem enkripsi dan *split*

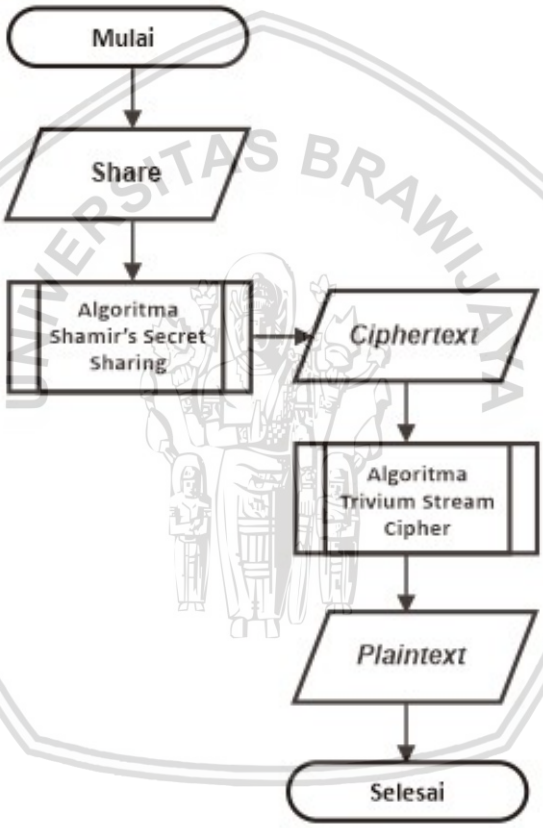
Berikut merupakan perancangan sistem enkripsi dan *split* dalam bentuk *flowchart* yang akan digunakan untuk menjelaskan bagaimana nilai yang akan diproses pada sistem ini, yakni di awali dengan *user* memasukkan data berupa *file PDF*, kemudian akan di enkripsi menggunakan algoritme Trivium Stream Cipher, kemudian akan menjadi data *cipher*, dari data *cipher* yang sudah ada akan di pecah menggunakan algoritme Shamir's Secret Sharing. Gambaran sistem enkripsi dan share bisa di lihat berdasarkan Gambar 4.1.



Gambar 4.1 Gambar Sistem Enkripsi dan *Share*

4.1.2 Sistem *Reconstruct* dan Dekripsi

Berikut merupakan perancangan sistem enkripsi dan share dalam bentuk *flowchart* yang akan digunakan untuk menjelaskan bagaimana nilai yang akan diproses pada sistem ini, yakni dimulai *user* meng*input*kan *share* , kemudian akan di *reconstruct* menggunakan algoritme Shamir's Secret Sharing, kemudian akan menjadi data *cipher* yang awal, kemudian data *cipher* itu akan di dekripsi menggunakan algoritme Trivium Stream Cipher, kemudian akan menjadi data awal. Gambaran sistem *reconstruct* dan dekripsi bisa dilihat berdasarkan Gambar 4.2.

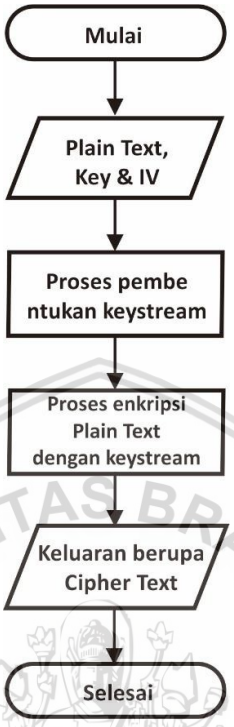


Gambar 4.2 Gambar Sistem *Reconstruct* dan Dekripsi

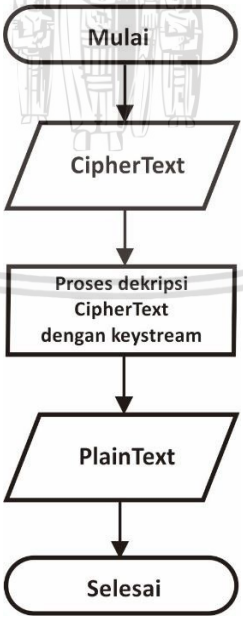
4.1.3 Sistem Algoritme Trivium Stream Cipher

Berikut merupakan perancangan sistem algoritme Trivium Stream Cipher dalam bentuk *flowchart* yang akan digunakan untuk menjelaskan bagaimana nilai yang akan diproses pada sistem ini, dalam sistem ini terdapat 2 proses yakni enkripsi dan dekripsi. Untuk yang pertama adalah proses enkripsi dimulai dari *user* memasukkan data, *key* dan *IV*. Kemudian dilanjutkan pada proses pembentukan *keystream*, setelah itu di lanjutkan pada proses enkripsi data menggunakan *keystream* yang di *XOR*kan dengan bit-bit data, kemudian sistem akan

menghasilkan data *cipher* atau data hasil enkripsi. Alur proses enkripsi dapat dilihat pada Gambar 4.3.



Gambar 4.3 Enkripsi



Gambar 4.4 Dekripsi

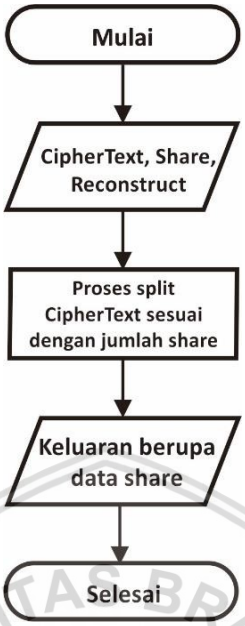
Proses yang kedua adalah deskripsi dapat dilihat pada Gambar 4.4, proses ini dimulai dari *user* memasukkan data *cipher*, kemudian proses dekripsi data

mengunakan *keystream* yang sama dengan proses enkripsi tadi, kemudian sistem akan menghasilkan data awal atau data asli sebelum ter enkripsi. Dari alur sistem enkripsi dan dekripsi ini dapat di manualisasikan seperti langkah-langkah dibawah ini. Adapun langkah-langkahnya adalah sebagai berikut :

1) Untuk proses pertama adalah enkripsi yang dimulai dari memasukkan *key* dan *IV* , disini saya inputkan *key* "matahari" yang dirumah ke bentuk biner menjadi :
   "011011010110000101110100011000010110100001100001011100100 1101001".
   sedangkan *IV* "bersinar" yang dirubah kebentuk biner menjadi :
   "011000100110010101110010011100110110100101101110011000010 1110010".
   Kemudian akan di proses sehingga menghasilkan *keystream* sebagai berikut :
   "110110101000011011010110100000100011000101010110110001000 0101101110111111100101100011100".
2) Kemudian memasukkan data berupa *file*, disini saya memasukkan *file pdf* dengan isi berupa teks "Skripsi" yang akan dirubah menjadi biner untuk proses enkripsi, setelah file di*convert* ke biner maka akan menghasilkan biner sebagai berikut :
   "011100110110101101110010011010010111000001110011011010010 0001010001000000000101000100000".
3) Kemudian adalah proses enkripsi yakni dengan meng *xor* kan antara biner *keystream* dengan biner yang di dapatkan dari *file*, sehingga menghasilkan *ciphertext* sebagai berikut :
   "101010011110110110100100111010110100000100100101101011010 0111001100111111101111101011100".
4) Sedangkan untuk proses dekripsi adalah dengan meng *xor* kan antara biner *keystream* dengan biner *ciphertext* yang dihasilkan dari proses enkripsi sehingga akan mengasilkan *plaintext* :
   "011100110110101101110010011010010111000001110011011010010 0001010001000000000101000100000".

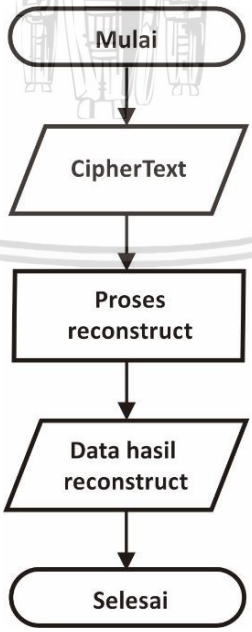
4.1.4 Sistem Algoritme Shamir's Secret Sharing

Berikut merupakan perancangan sistem algoritme Shamir's Secret Sharing dalam bentuk *flowchart* yang akan di gunakan untuk menjelaskan bagaimana nilai yang akan diproses pada sistem ini, pada sistem ini terdapat dua proses yakni proses *share* dan proses *reconstruct* . Untuk proses yang pertama adalah *split* yakni dimulai *user* meng*input*kan data *cipher*, jumlah *share*, dan jumlah *reconstruct*, kemudian mulailah proses *split* data sesuai dengan jumlah *share* yang dimasukkan oleh *user,* kemudian yang terakhir adalah keluar hasil berupa data *share*. Alur proses *split* dapat dilihat pada Gambar 4.5.



Gambar 4.5 *Split*

Untuk proses yang kedua adalah *reconstruct*, proses ini digunakan untuk mengembalikan data yang sudah di *split* menjadi data awal kembali dengan jumlah data *reconstruct* yang sesuai dengan masukan *user*. Proses ini dimulai dari *user* memasukkan data *share* sesuai dengan keinginan *user*, kemudian dilanjutkan pada proses *reconstruct* data, kemudian yang terakhir adalah sistem memberikan hasil keluaran berupa data hasil *reconstruct* yang berupa data *cipher*. Gambar *reconstruct* dapat di lihat pada Gambar 4.6.



Gambar 4.6 *Reconstruct*

Dari alur sistem *split* dan *reconstruct* ini dapat di manualisasikan seperti langkah-langkah dibawah ini. Adapun langkah-langkahnya adalah sebagai berikut :

1) Untuk yang pertama adalah merubah biner dari *ciphertext* yang di peroleh dari proses enkripsi menjadi desimal untuk biner adalah : "01110011011010110111001001101001011100000111001101101001000010100010000000001010001000000". Dan dirubah menjadi desimal menjadi "205430706589212422649671596".
2) Setelah itu akan di bangun bilangan prima secara acak dengan batasan harus lebih besar dari bilangan desimal *ciphertext* "591288252727333192900453463" yang nantinya digunakan untuk proses *split*.
3) Kemudian akan di bangun koefisien yang jumlahnya bilangan *reconstruct* dikurangi 1. Sehingga jika *reconstruct*nya adalah 3, maka koefisienya adalah 2. Koefisien ini digunakan untuk proses *split*.
    a) 37420750316187504540944668
    b) 116526318962867746471701177
4) Kemudian dari desimal *ciphertext* tersebut akan dipecah menjadi 3 bagian. Sehingga akan dihasilkan 3 macam bilangan desimal.
    a) 96767668911168125583577344
    b) 33169013887001958087912754
    c) 10056211441691713777585897
5) Kemudian untuk menggabungkan kembali pada *file* awal atau bilangan desimal awal, hanya diperlukan 2 dari 3 bagian file tersebut. Bilangan yang diperlukan yakni
    a) 96767668911168125583577344
    b) 33169013887001958087912754

    Dari gabungan 2 pecahan ini akan menghasilkan bilangan desimal yang sesuai dengan bilangan desimal ciphertext yakni : "205430706589212422649671596".

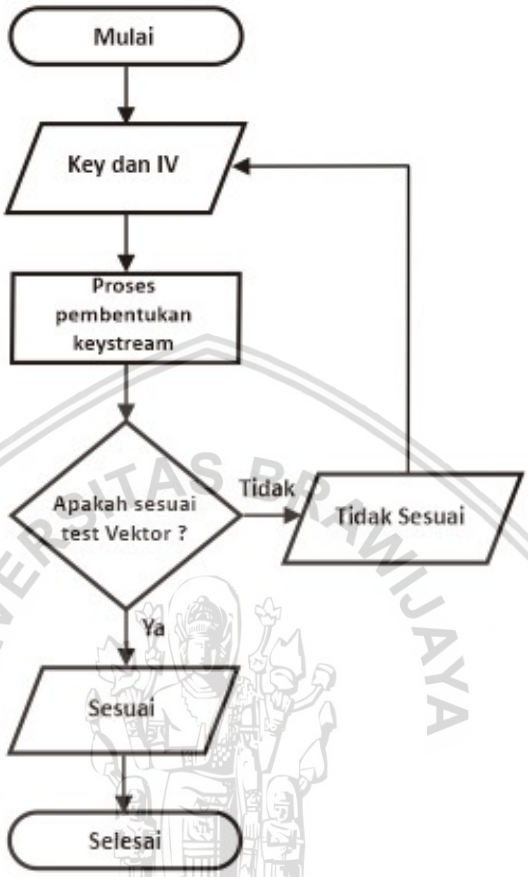
4.1.5 Sistem Pengujian

Berikut merupakan perancangan sistem pengujian dalam bentuk *flowchart* yang akan digunakan untuk menjelaskan bagaimana nilai yang akan diproses pada sistem ini. Dalam sistem pengujian terdapat empat proses pengujian, yakni adalah pengujian *test vector*, pengujian enkripsi dan dekripsi, pengujian *split* dan *reconstruct*, pengujian waktu.

4.1.5.1 Pengujian *Test Vector*

Sistem pengujian ini bertujuan untuk validasi bahwa algoritme Trivium Stream Cipher yang di terapkan sudah sesuai dengan algoritme Trivium Stream Cipher yang dibuat oleh Christophe De Canni're dan Bart Preneel. Proses pengujian ini di mulai dari user memasukkan *key* dan *IV* , kemudian akan diproses dalam pembentukan *keystream*, dan dari *keystream* ini lah yang akan di cocokkan dengan nilai *keystream test vector*. Jika *test vector* tidak cocok, maka akan

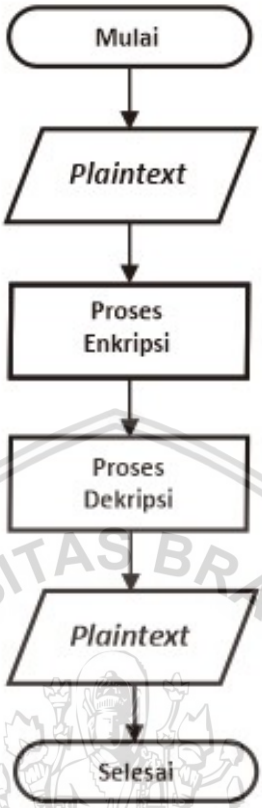
dikembalikan ke proses input *key* dan *IV* Gambar pengujian *test vector* dapat dilihat pada Gambar 4.7.



Gambar 4.7 Pengujian *Test Vector*

4.1.5.2 Pengujian *Confidentiality*

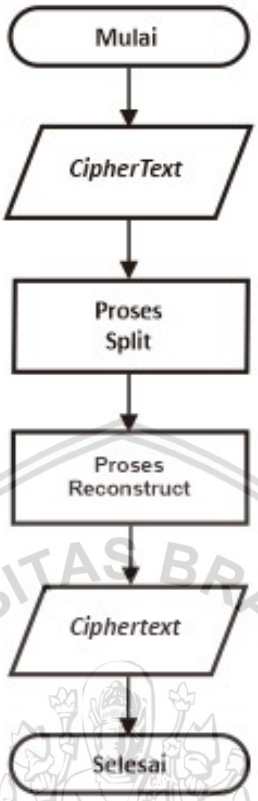
Sistem pengujian ini bertujuan untuk mengetahui apakah algoritme yang menggunakan teknik enkripsi dan dekripsi sudah berjalan dengan baik. Pengujian enkripsi bisa di anggap berhasil jika data yang diamankan dapat berubah menjadi data acak yang sulit dibaca. Pengujian dekripsi bisa dianggap berhasil jika data acak dapat kembali menjadi data semula. Cara pengujian ini dilakukan dengan melihat hasil enkripsi dan menyesuaikan data sebelum proses enkripsi dengan data setelah proses dekripsi. Proses pengujian ini dimulai dari *user* memasukkan data, kemudian akan diproses dalam sistem enkripsi dan *secret sharing*, kemudian akan di cocokkan hasil dari kedua sistem tersebut apakah sesuai dengan data awal yang dimasukkan. Alur sistem enkripsi dan dekripsi dapat dilihat pada Gambar 4.8.



Gambar 4.8 Sistem Enkripsi dan Dekripsi

4.1.5.3 Pengujian *Availability*

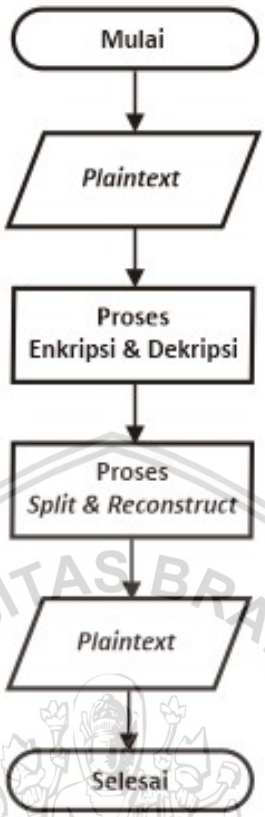
Sistem pengujian ini bertujuan untuk mengetahui apakah algoritme Shamir's Secret Sharing yang digunakan sudah berjalan dengan baik. Pengujian *split* bisa di anggap berhasil jika data yang dipecah dapat di kembalikan ke pada data awal dengan jumlah *reconstruct* yang berbeda-beda. Proses pengujian ini dimulai dari *user* memasukkan data, kemudian akan diproses dalam sistem *secret sharing*, kemudian akan di cocokkan apakah data hasil dari sistem sudah sesuai dengan data awal masukan dari *user*, jika data sudah sama berarti bisa dianggap berhasil. Data awal yang dimasukkan adalah *ciphertext*. Alur pengujian *split* dan *reconstruct* dapat dilihat pada Gambar 4.9.



Gambar 4.9 Pengujian *Split* dan *Reconstruct*

4.1.5.4 Pengujian Kevalidan *File*

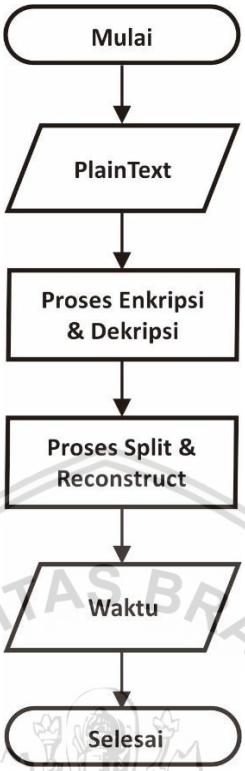
Sistem pengujian ini bertujuan untuk Pengujian ini bertujuan untuk mengetahui *file* yang di hasilkan oleh proses Shamir's Secret Sharing dan Trivium Stream Cipher dapat sesuai dengan *file* yang asli. Proses pengujian ini dimulai dari user memasukkan data, kemudian diproses pada sistem enkripsi dan *secret sharing*, kemudian akan di cocokkan apakah *file* hasil dari kedua sistem telah sesuai dengan *file* asli. Jika sesuai maka bisa di anggap berhasil. Alur pengujian kevalidan *file* bisa dilihat pada Gambar 4.10.



Gambar 4.10 Pengujian Kevalidan *File*

4.1.5.5 Pengujian Waktu

Sistem pengujian ini bertujuan untuk mengetahui waktu yang dibutuhkan pada proses enkripsi, dekripsi, *split* dan *reconstruct file PDF*. Proses ini di awali dari *user* memasukkan data, kemudian akan di proses dalam sistem enkripsi dan *secret sharing*, kemudian akan menghasilkan *output file* olahan beserta waktu yang di butuhkan selama proses pengolahan tersebut. Waktu yang telah dihasilkan nantinya akan di bandingkan sesuai dengan ukuran file yang berbeda-beda. Alur pengujian waktu dapat dilihat pada Gambar 4.11.



Gambar 4.11 Pengujian Waktu

BAB 5 MPLEMENTASI

Pada bab sebelumnya kita sudah membahas konsep perancangan, yang mana akan menjadi dasar dalam melakukan implementasi algoritme Trivium Stream Cipher dan Shamir's Secret Sharing. Didalam tahap ini akan dijelaskan bagaimana algoritme ini diterapkan dalam bahasa pemograman java.

5.1 Algoritme Trivium Stream Cipher

Pada algoritme Trivium Stream Cipher terdapat beberapa *source code* yang sudah di program sebagaimana perancangan sistem yang telah tertera pada bab sebelumnya, diantaranya adalah *source code initial state, source code keystream, source code enkripsi, sourcode dekripsi, source code convert string to binary, source code* mengambil bit dari *file PDF*.

5.1.1 Source Code Initial State

Pada algoritme Trivium Stream Cipher terdapat *source code initial state* yang berfungsi sebagai tahapan awal dalam algoritme Trivium Stream Cipher. Pada tahapan ini menggunakan tiga *input*an yakni *key* sepanjang 80 bit, *IV* sepanjang 80 bit, *plain* panjangnya disesuaikan dengan inputan *user,* dari *key* dan *IV* di gunakan untuk membentuk *shift register* sepanjang 288 bit. Jika panjang *key* dan *IV* lebih dari 80 bit, maka akan di katakan *false* dan sistem tidak akan berjalan namun apabila kurang dari 80 bit, sistem akan tetap berjalan. *Plain* di gunakan untuk mengetahui panjang bit dari *plain* tersebut yang nantinya di gunakan untuk menentukan panjang dari *keystream* algoritme Trivium Stream Cipher. Tabel *source code initial state* dapat di lihat pada tabel berikut ini.

Algoritme 1 : Fungsi InitialState

```
1   public void InitialState(String key, String IV, String plain) throws
2   IOException {
3           Converter conv = new Converter();
4           startTime = Sistem.currentTimeMillis();;
5           String key_result = conv.stringToBinary(key);
6           if (key_result.length() > 80) {
7               a = true;
8           }
9           String IV_result = conv.stringToBinary(IV);
10          if (IV_result.length() > 80) {
11              b = true;
12          }
13          String plain_result = conv.stringToBinary(plain);
14          int jum_IV = IV_result.length() + 93;
15          if (a == false && b == false) {
16              for (int k = 0; k < key_result.length(); k++) {
17                  sr[k]                                                    =
18  Byte.parseByte(String.valueOf(key_result.charAt(k)));
19              }
20              if (key_result.length() < 80) {
21                  for (int k = key_result.length(); k < 80; k++) {
22                      sr[k] = 0;
23                  }
24              }
25              for (int k = 80; k <= 92; k++) {
26                  sr[k] = 0;
27              }
28              for (int k = 93; k < jum_IV; k++) {
```

```
29                  sr[k]                                                   =
30  Byte.parseByte(String.valueOf(IV_result.charAt(k - 93)));
31              }
32              if (IV_result.length() < 80) {
33                  for (int k = IV_result.length(); k < 80; k++) {
34                      sr[k + 93] = 0;
35                  }
36              }
37              for (int k = 173; k <= 176; k++) {
38                  sr[k] = 0;
39              }
40              for (int k = 177; k <= 284; k++) {
41                  sr[k] = 0;
42              }
43              for (int k = 285; k <= 287; k++) {
44                  sr[k] = 1;
45              }
46              Sistem.out.print("Plain Text :");
47              for (int i = 0; i < plain_result.length(); i++) {
48  Sistem.out.print(Byte.parseByte(String.valueOf(plain_result.charAt
49  (i))));
50              }
51              Sistem.out.println("");
52              Keystream(plain_result);
53          } else if (a == true && b == true) {
54              Sistem.out.println("the Key And IV is too long ...");
55          } else if (b == true) {
56              Sistem.out.println("the IV is too long ...");
57          } else if (a == true) {
58              Sistem.out.println("the Key is too long ...");
59          }
60      }
```

5.1.2 Source Code Keystream

Pada algoritme Trivium Stream Cipher terdapat *source code keystream* yang berfungsi sebagai pembangun bit-bit *keystream* yang panjangnya sesuai dengan *plaintext* yang telah di *input*kan oleh *user.* Pembangunan bit-bit *keystream* di lakukan dengan cara melakukan *loop* (perputaran) sebanyak 288 x 4 atau 1152, di tambah dengan panjang dari *plaintext* itu sendiri. Aturanya adalah apabila sudah mencapai 1152 loop (putaran), maka *keystream* boleh diambil sesuai dengan panjang *plaintext* atau lebih. *Keystream* ini di gunakan untuk mengenkripsi *plaintext* dengan cara meng *XOR* kan antara bit *keystream* dengan bit *plaintext*. Tabel *source code keystream* dapat dilihat pada tabel berikut ini.

Algoritme 2 : Fungsi pembentukan keystream

```
1   public void Keystream(String input) throws IOException {
2           Sistem.out.print("Keystream :");
3           for (int i = 0; i < 1152 + input.length(); i++) {
4               t1 = sr[65] ^ sr[92];
5               t2 = sr[161] ^ sr[176];
6               t3 = sr[242] ^ sr[287];
7               if (i > 1151) {
8                   z = t1 ^ t2 ^ t3;
9                   Sistem.out.print(z);
10                  keystream[i - 1152] = z;
11              }
12              t1 = t1 ^ (sr[90] & sr[91]) ^ sr[170];
13              t2 = t2 ^ (sr[174] & sr[175]) ^ sr[263];
14              t3 = t3 ^ (sr[285] & sr[286]) ^ sr[68];
15              for (int j = 287; j > 0; j--) {
16                  sr[j] = sr[j - 1];
```

```
17              }
18              sr[0] = t3;
19              sr[93] = t1;
20              sr[177] = t2;
21          }
22          Sistem.out.println("");
23          Encrypt(input);
24      }
```

5.1.3 *Source Code* Enkripsi

Pada algoritme Trivium Stream Cipher terdapat *source code* enkripsi yang berfungsi untuk mengenkripsi *plaintext* yang telah di *input*kan oleh *user.* Proses enkripsi dilakukan dengan cara meng *XOR* kan antara bit-bit dari *keystream* dengan bit-bit dari *plaintext* itu sendiri dan akan menjadi *ciphertext*. Tabel *source code* enkripsi dapat di lihat pada tabel berikut ini.

Algoritme 3 : Fungsi Enkripsi

```
1   public void Encrypt(String input) throws IOException {
2           startTime = Sistem.currentTimeMillis();
3           Scanner scn = new Scanner(Sistem.in);
4           Converter conv = new Converter();
5           final int CERTAINTY = 300;
6           final SecureRandom random = new SecureRandom();
7           String result = input;
8           String result_string = "";
9           Byte[] result_array = new Byte[result.length()];
10          //Sistem.out.println("Panjang Byte : " + result.length());
11          Byte[] result_xor_array = new Byte[result.length()];
12
13          for (int i = 0; i < result_array.length; i++) {
14              result_array[i]                                          =
15  Byte.parseByte(String.valueOf(result.charAt(i)));
16              result_xor_array[i]    =    (byte)    (keystream[i]    ^
17  result_array[i]);
18              result_string += result_xor_array[i];
19              binarykeystream += keystream[i];
20          }
21
22  Sistem.out.println("=====================================");
23          Sistem.out.println("Cipher Text :" + result_string);
24          try {
25              File            newTextFile            =            new
26  File("E:/Test/FileEnkripsi/Enkripsi.txt");
27              try (FileWriter fw = new FileWriter(newTextFile)) {
28                  fw.write(conv.BinToAsCII(result_string));
29              }
30          } catch (IOException iox) {
31          }
32          Sistem.out.print("Dirubah Ke Decimal :");
33          BigDecimal a2;
34          a2 = conv.bitStringToBigDecimal(result_string);
35          Sistem.out.println(a2);
36          endTime = Sistem.currentTimeMillis();
37          NumberFormat formatter = new DecimalFormat("#0.00000");
38          Sistem.out.print("Execution   time   to   Encryp   is   "   +
39  formatter.format((endTime - startTime) / 1000d) + " seconds");
40          Sistem.out.println("");
```

5.1.4 *Source Code* Dekripsi

Pada algoritme Trivium Stream Cipher terdapat *source code* dekripsi yang berfungsi untuk mendekripsi atau mengembalikan ke bentuk semula dari *plaintext*

yang sudah terenkripsi. Proses dekripsi pada Trivium Stream Cipher di lakukan dengan cara meng *XOR* kan kembali dari bit-bit *keystream* dengan bit-bit dari *ciphertext*, hasil dari proses ini adalah *plaintext* atau bentuk semula. Tabel *source code* dekripsi dapat di lihat pada tabel berikut ini.

Algoritme 4 : Fungsi Dekripsi

```
1   public void Decrypt(String keystream, String cipher) {
2           Converter fc = new Converter();
3           Byte[] a_array = new Byte[cipher.length()];
4           Byte[] b_array = new Byte[keystream.length()];
5           String plain_binary = "";
6           String plain = "";
7           Byte[] hasil = new Byte[cipher.length()];
8
9           for (int i = 0; i < a_array.length; i++) {
10              a_array[i]                                          =
11  Byte.parseByte(String.valueOf(cipher.charAt(i)));
12
13          }
14
15          for (int i = 0; i < b_array.length; i++) {
16              b_array[i]                                          =
17  Byte.parseByte(String.valueOf(keystream.charAt(i)));
18
19          }
20
21          for (int i = 0; i < cipher.length(); i++) {
22              hasil[i] = (byte) (b_array[i] ^ a_array[i]);
23              plain_binary += hasil[i];
24
25          }
26          Sistem.out.println("Dekripsi = " + plain_binary);
27          String decbyt = fc.BinToAsCII(plain_binary);
28          FileTest.writeUsingIText(decbyt);
29      }
```

5.1.5 Source Code Convert String to Binary

Pada algoritme Trivium Stream Cipher terdapat *source code convert string to binary* yang di gunakan untuk mengonvert inputan *key,IV* dan *plain* berupa *string*. Perubahan ini dari *string* ke *binary* yang nantinya akan di gunakan dalam proses selanjutnya yakni *initial state*, enkripsi, dan dekripsi. *Source code* dapat dilihat pada tabel berikut ini.

Algoritme 5 : Fungsi convert string to binary

```
1   public String stringToBinary(String string) {
2           String result = "";
3           String tmpStr;
4           int tmpInt;
5           char[] messChar = string.toCharArray();
6           for (int i = 0; i < messChar.length; i++) {
7               tmpStr = Integer.toBinaryString(messChar[i]);
8
9               tmpInt = tmpStr.length();
10
11              if (tmpInt != 8) {
12                  tmpInt = 8 - tmpInt;
13                  if (tmpInt == 8) {
14                      result += tmpStr;
15                  } else if (tmpInt > 0) {
16                      for (int j = 0; j < tmpInt; j++) {
17                          result += "0";
```

```
18
19                      }
20                  } else {
21                      Sistem.out.println("arguments bits to small
22  ...");
23                  }
24              }
25              result += tmpStr;
26              //
27          }
28          result += "";
29          return result;
30      }
```

5.1.6 Source Code Read File

Pada algoritme Trivium Stream Cipher terdapat *source code readfile*, sebenanrnya *source code* ini merupakan tambahan yang di sesuaikan dengan *input*an *user*, di karenakan dalam penelitian ini menggunakan objek *file* maka dibutuhkan lah *source code readfile* ini. *Source code readfile* ini berfungsi untuk membaca bit dari *file* tersebut, yang nantinya akan digunakan untuk *input*an sebagai *plaintext* pada algoritme Trivium Stream Cipher. *Source code* dapat di lihat pada tabel berikut ini.

Algoritme 6 : Fungsi read file

```
1   public static String readfile(String name){
2           String textFromPage = null;
3           String FILE_NAME = name;
4           PdfReader reader;
5
6           try {
7
8               reader = new PdfReader(FILE_NAME);
9
10              // pageNumber = 1
11              textFromPage                                          =
12  PdfTextExtractor.getTextFromPage(reader, 1);
13              //writeUsingIText(textFromPage);
14              //Sistem.out.println("DOne");
15
16              reader.close();
17
18          } catch (IOException e) {
19              e.printStackTrace();
20          }
21          return textFromPage;
22      }
```

5.2 Algoritme Shamir's Secret Sharing

Pada algoritme Shamir's Secret Sharing terdapat beberapa *source code* yang sudah di program sebagaimana perancangan sistem yang telah tertera pada bab sebelumnya, diantaranya adalah *source code split, source code combine, source code* menulis bit ke bentuk file dokumen berformat **.pdf*, *source code* menyimpan share ke laptop, *source code* mengambil *share* dari laptop, *source code convert* biner ke desimal, *source code convert* desimal ke biner, *source code convert* biner ke kode *ASCII* .

5.2.1 Source Code Split

Pada algoritme Shamir's Secret Sharing terdapat *source code split* yang berfungsi sebagai pemecah *secret key* yang di inginkan dan menghasilkan *share key*. Tujuan pemecahan ini adalah untuk menambah *availability* dari *secret key* tersebut, apabila nanti ada sebagian sebagian *secret key* yang hilang akan mudah untuk me*reconstruct secret key* awal dengan share yang tersisa. Dalam *source code* split ini di syaratkan menggunakan bilangan prima untuk proses *mod,* maka dari itu dalam *source code* ini menggunakan *library import java.util.random* untuk membangun bilangan prima secara *random*. Kemdian file hasil *split* akan disimpan dalam file *txt. Inputan yang digunakan adalah nilai dari secret, jumlah *reconstruct*, jumlah *share* , bilangan prima, dan fungsi *random*. Tabel *source code split* dapat dilihat pada tabel berikut ini.

Algoritme 7 : Fungsi Split/SecretShare

```
1   public static SecretShare[] split(final BigInteger secret, int
2   needed, int available, BigInteger prime, Random random) {
3           Sistem.out.println("Prime Number: " + prime);
4
5           final BigInteger[] coeff = new BigInteger[needed];
6           coeff[0] = secret;
7           for (int i = 1; i < needed; i++) {
8               BigInteger r;
9               while (true) {
10                  r = new BigInteger(prime.bitLength(), random);
11                  if (r.compareTo(BigInteger.ZERO) > 0 &&
12  r.compareTo(prime) < 0) {
13                      break;
14                  }
15              }
16              coeff[i] = r;
17          }
18
19          final SecretShare[] shares = new SecretShare[available];
20          for (int x = 1; x <= available; x++) {
21              BigInteger accum = secret;
22              for (int exp = 1; exp < needed; exp++) {
23              accum=
24  accum.add(coeff[exp].multiply(BigInteger.valueOf(x).pow(exp).mod(p
25  rime))).mod(prime);
26              }
27              shares[x - 1] = new SecretShare(x, accum);
28              //Write File To TXT
29
30              String str2 = accum.toString();
31              FileTest pdf = new FileTest();
32              writeShareUsingIText(str2,x);
33
34              try {
35                  String str = accum.toString();
36
37  public SecretShare(final int number, final BigInteger share) {
38          this.number = number;
39          this.share = share;
40      }
41
42      public int getNumber() {
43          return number;
44      }
45
46      public BigInteger getShare() {
```

```
47          return share;
48      }
49
50      @Override
51      public String toString() {
52          return "SecretShare [num=" + number + ", share=" + share +
53  "]";
54      }
55
56      prIVate final int number;
57      prIVate final BigInteger share;
58                  File newTextFile = new
59  File("G:/Test/Share/share_ke_" + x +  ".txt");
60                  FileWriter fw = new FileWriter(newTextFile);
61                  fw.write(str);
62                  //writeShareUsingIText(str,x);
63                  fw.close();
64
65              } catch (IOException iox) {
66                  //do stuff with exception
67                  iox.printStackTrace();
68              }
69
70              Sistem.out.println("Share " + shares[x - 1]);
71          }
72          return shares;
73      }
```

5.2.2 Source Code Combine

Pada algoritme shamir's secret sharing terdapat *source code combine* yang berfungsi untuk menggabungkan *share key* yang telah di bangun pada *source code split*. Proses penggabungan ini tidak melibatkan keseluruhan share key yang dibangun, namun cukup membatasi dengan sebagian dari share key yang dibangun. Inputan yang digunakan adalah *share* yang telah dipilih, bilangan prima *random*. Tabel source code combine dapat dilihat pada tabel berikut ini.

Algoritme 8 : Fungsi Combine

```
1   public static BigInteger combine(final SecretShare[] shares, final
2   BigInteger prime) {
3           BigInteger accum = BigInteger.ZERO;
4
5           for (int formula = 0; formula < shares.length; formula++)
6   {
7               BigInteger numerator = BigInteger.ONE;
8               BigInteger denominator = BigInteger.ONE;
9
10              for (int count = 0; count < shares.length; count++) {
11                  if (formula == count) {
12                      continue; // If not the same value
13                  }
14                  int startposition = shares[formula].getNumber();
15                  int nextposition = shares[count].getNumber();
16
17                  numerator =
18  numerator.multiply(BigInteger.valueOf(nextposition).negate()).mod(
19  prime); // (numerator * -nextposition) % prime;
20                  denominator =
21  denominator.multiply(BigInteger.valueOf(startposition-
22  nextposition)).mod(prime); // (denominator * (startposition-
23  nextposition)) % prime;
24              }
25              BigInteger value = shares[formula].getShare();
```

```
26              BigInteger tmp =
27  value.multiply(numerator).multiply(modInverse(denominator,
28  prime));
29              accum = prime.add(accum).add(tmp).mod(prime); //
30  (prime + accum + (value * numerator * modInverse(denominator))) %
31  prime;
32          }
33          Sistem.out.println("The secret is: " + accum);
34
35          return accum;
36      }
37  prIVate static BigInteger[] gcdD(BigInteger a, BigInteger b) {
38          if (b.compareTo(BigInteger.ZERO) == 0) {
39              return new BigInteger[]{a, BigInteger.ONE,
40  BigInteger.ZERO};
41          } else {
42              BigInteger n = a.dIVide(b);
43              BigInteger c = a.mod(b);
44              BigInteger[] r = gcdD(b, c);
45              return new BigInteger[]{r[0], r[2],
46  r[1].subtract(r[2].multiply(n))};
47          }
48      }
49
50      prIVate static BigInteger modInverse(BigInteger k, BigInteger
51  prime) {
52          k = k.mod(prime);
53          BigInteger r = (k.compareTo(BigInteger.ZERO) == -1) ?
54  (gcdD(prime, k.negate())[2]).negate() : gcdD(prime, k)[2];
55          return prime.add(r).mod(prime);
56      }
```

5.2.3 Source Code Menulis bit ke Bentuk *File*

Pada penelitian yang saya buat hasil dari proses *combine* akan ditulis ke bentuk *file* dokumen dengan format *PDF* dikarenakan file awal yang di enkripsi adalah file dokumen dengan format *PDF*. Penulisan bit-bit *file* menggunakan *library import com.itextpdf.text.*, import com.itextpdf.text.pdf.PdfWriter.* Tabel source code menlis bit ke bentuk file dapat dilihat pada tabel berikut ini.

```
Algoritme 9 : Fungsi write bit to file
1   public static void writeUsingIText(String h) {
2           String FILE_NAME = "E:/Test/FileHasil/3.pdf";
3           //String h = byt;
4           Document document = new Document();
5           Rectangle one = new Rectangle(613, 1000);
6           document.setPageSize(one);
7   //          document.newPage();
8           try {
9               PdfWriter.getInstance(document,                    new
10  FileOutputStream(new File(FILE_NAME)));
11              document.setMargins(72, 0, 60, 0);
12              document.open();
13              Font f1 = FontFactory.getFont(FontFactory.TIMES_ROMAN,
14  11);
15              Paragraph p = new Paragraph(22, h, f1);
16              //document.add(new Paragraph(h, f1));
17              document.add(p);
18              document.close();
19              Sistem.out.println("Done");
20          } catch (FileNotFoundException | DocumentException e) {
21              e.printStackTrace();
22          } catch (IOException e) {
```

```
23              e.printStackTrace();
24          }
        }
```

5.2.4 *Source Code* Menyimpan *Share* ke Laptop

Pada penelitian yang saya buat hasil dari proses *split* akan ditulis ke dalam *file*. Proses ini akan berulang sesuai dengan jumlah *share* yang dibuat oleh *user*. Format yang akan dipakai saat penulisan adalah **.txt*. File ini nanti akan diproses kembali dalam proses *combine*. Dalam *source code* ini menggunakan *library import java.io.FileWriter.* Tabel *source code* menyimpan *share* ke laptop dapat di lihat pada tabel berikut ini.

```
Algoritme 10 : Fungsi menyimpan share
1   try {
2                   String str = accum.toString();
3                   File          newTextFile          =          new
4   File("E:/Test/Share/share_ke_" + x + ".txt");
5                   FileWriter fw = new FileWriter(newTextFile);
6                   fw.write(str);
7                   //writeShareUsingIText(str,x);
8                   fw.close();
9
10              } catch (IOException iox) {
11                  //do stuff with exception
12                  iox.printStackTrace();
13              }
```

5.2.5 *Source Code* Mengambil *Share* Dari Laptop

Pada penelitian yang saya buat setelah *share key* dalam proses *split* tersimpan dalam *file* berformat **.txt* maka akan diambil kembali untuk proses *combine* atau menggabungkan *share key* untuk menjadikan *secret key* yang awal. Dalam *souce code* ini menggunakan *library import java.io.FileReader*. Tabel *source code* mengambil *share* dari laptop dapat dilihat pada tabel berikut ini.

```
Algoritme 11 : Fungsi mengambil share
1   for (int i = 0; i < share; i++) {
2                       String c =
3   FileTest.readFile("E:/Test/Share/share_ke_" + (i + 1) + ".txt");
4                       BigInteger accum = new BigInteger(c);
5                       shares2[i] = new SecretShare((i + 1), accum);
6                   }
```

5.2.6 *Source Code Convert* Biner ke Desimal

Pada algoritme Shamir's Secret Sharing *input*anya adalah desimal, sedangkan hasil enkripsi adalah berbentuk biner, maka perlu di rubah dahulu hasil enkripsi dari biner ke desimal. Tabel *source code convert* biner ke desimal dapat dilihat pada tabel berikut ini.

```
Algoritme 12 : Fungsi convert biner to decimal
1   public BigDecimal bitStringToBigDecimal(String bitStr) {
2
3           BigDecimal sum = new BigDecimal("0");
4           BigDecimal base = new BigDecimal(2);
5           BigDecimal temp;
6           for (int i = 0; i < bitStr.length(); i++) {
7               if (bitStr.charAt(i) == '1') {
```

```
8                   int exponent = bitStr.length() - 1 - i;
9                   temp = base.pow(exponent);
10                  sum = sum.add(temp);
11              }
12
13          }
14          return sum;
15      }
```

5.2.7 *Source Code Convert* Desimal ke Biner

Pada algoritme Shamir's Secret Sharing jika *input*anya adalah desimal maka *output*nya pun desimal, maka untuk melanjutkan ke tahap dekripsi yang membutuhkan *input*an biner perlu adanya perubahan dari hasil algoritme Shamir's Secret Sharing yang berupa desimal, kita rubah ke dalam bentuk biner. Tabel *source code convert* desimal ke biner dapat di lihat pada tabel berikut ini.

Algoritme 13 : Fungsi conver decimal to biner

```
1   public String integerToBinary(String s) {
2           if (!s.matches("[0-9]+")) {
3               throw new IllegalArgumentException(s + " cannot be
4   converted to integer");
5           }
6
7           String result = "";
8           while (!s.equals("0") && !s.equals("1")) {
9               int lastDigit =
10  Character.getNumericValue(s.charAt(s.length() - 1));
11              result = lastDigit % 2 + result; //if last digit is
12  even prepend 0, otherwise 1
13              s = dIVideByTwo(s);
14          }
15          return (s + result).replaceAll("^0*", "");
16      }
```

5.2.8 Source Code Convert Biner ke Kode *ASCII*

Pada penelitian yang saya buat setelah proses dekripi maka bit-bit biner akan ditulis kembali kedalam bentuk *file* dokumen berformat *PDF*. Tabel *source code convert* biner ke kode *ASCII* dapat dilihat pada tabel berikut ini

Algoritme 14 : Fungsi convert biner to ASCII

```
1   public String BinToAsCII(String inp) {
2           String input = inp;
3           String output = "";
4           for (int i = 0; i <= input.length() - 8; i += 8) {
5               int k = Integer.parseInt(input.substring(i, i + 8), 2);
6               output += (char) k;
7           }
8           return output;
9       }
```

BAB 6 PENGUJIAN DAN PEMBAHASAN

6.1 Parameter Pengujian

Parameter yang digunakan untuk menguji sistem ini adalah :

1. Pengujian pada algoritme Trivium Stream Cipher menggunakan 2 cara yakni:
    a. pengujian validitas test vector
    b. pengujian enkripsi dan deskripsi (*confidentiality*).
2. Pengujian pada algoritme Shamir's Secret Sharing dengan cara *split* dan *reconstruct* (*availability*).
3. Pengujian validitas *file* setelah diterapkan algoritme Trivium Stream Cipher dan Shamir's Secret Sharing.
4. Pengujian waktu yang dibutuhkan pada proses pembentukan *keystream, enkripsi, dekripsi, split*, *reconstruct* dan total waktu keseluruhan sistem .

6.2 Pengujian Validitas *Test Vector*

6.2.1 Tujuan Pengujian

Tujuan dari pengujian ini adalah untuk mengetahui kesesuain keluaran *keystream* yang dihasilkan oleh sistem dengan *test vector* dari pembuat algoritme *trivium stream cipher*. Jika sudah sesuai maka *keystream* yang di hasilkan oleh sistem bisa di katakan valid.

6.2.2 Prosedur Pengujian

Pengujian validitas akan di lakukan dengan mengambil 4 *test vector* yang sudah tersedia dari paper Trivium Stream Cipher, proses ini akan di mulai setelah *user* memasukkan *IV* dan *plaintext*. Setelah *keystream* diproses oleh sistem, maka *keystream* tersebut akan di cocokkan dengan *test vector*.

6.2.3 Hasil dan Pembahasan

Seperti yang di jelaskan pada sub bab tujuan pengujian validasi keluaran *keystream* pada sistem, pengujian akan mengambil 4 sampel data dan akan menyamakan hasil data sampel dengan *test vector*. Berikut hasil sampel validasi *keystream* pada algoritme Trivium Stream Cipher. Pada gambar 6.1 *test vector* algoritme Trivium Stream Cipher adalah hasil pengujian yang telah di tetapkan oleh pembuat algoritme Trivium Stream Cipher.

Test Vector

- Key=80000000000000000000
  IV=00000000000000000000
  Keystream=38EB86FF730D7A9CAF8DF13A4420540
- Key=00400000000000000000
  IV=00000000000000000000
  Keystream=61208D286BC1DC431171EDA5CAF79D95
- Key=00002000000000000000
  IV=00000000000000000000
  Keystream=C8F9031DABF8DB03FF120D05512B5F24
- Key=00000010000000000000
  IV=00000000000000000000
  Keystream=F7E523040E86EA2C46A2BE705BFC6259

Gambar 6.1 *Test Vector* Trivium Stream Cipher

Pada gambar 6.1 merupakan *test vector* yang diambil dari paper pembuat algoritme Trivium Stream Cipher dengan bentuk *hexadecimal*. Berikut tabel 6.1 hasil pengujian validitas.

Tabel 6.1 Pengujian Validitas *Test Vector*

No	*Key*	*IV*	*Keystream* Sistem	*Test Vector Trivium Stream Cipher*	Valid/ Tidak
1	0x80000000000000000000	0x00000000000000000000	0x38EB86FF730D7A9CAF8DF13A4420540	0x38EB86FF730D7A9CAF8DF13A4420540	Valid
2	0x00400000000000000000	0x00000000000000000000	0x61208D286BC1DC431171EDA5CAF79D95	0x61208D286BC1DC431171EDA5CAF79D95	Valid
3	0x00002000000000000000	0x00000000000000000000	0xC8F9031DABF8DB03FF120D05512B5F24	0xC8F9031DABF8DB03FF120D05512B5F24	Valid
4	0x00000010000000000000	0x00000000000000000000	0xF7E52304086EA2C46A2BE705BFC6259	0xF7E52304086EA2C46A2BE705BFC6259	Valid

Pada Tabel 6.1 menjelaskan bahwa *keystream* yang di hasilkan oleh sistem telah sesuai dengan *test vector* pada paper pembuat algoritme Trivium Stream Cipher. Sehingga pengujian *test vector* ini dapat dikatakan valid.

6.3 Pengujian *Confidentiality*

6.3.1 Tujuan Pengujian

Pengujian confidentiality dilakukan dengan cara enkripsi dan dekripsi untuk memastikan bahwa data di enkripsi sesuai dengan algoritme yang digunakan yakni Trivium Stream Cipher. Untuk memastikan ketepatanya , maka

perlu di cocokan nilai dari *plain text* pada saat sebelum di proses dengan *plain text* hasil dari proses dekripsi

6.3.2 Prosedur Pengujian

Prosedur pengujian yang di gunakan pada pengujian ini adalah dengan cara mencocokkan nilai *plaintext* yang di*input*kan dengan *plaintext* yang di hasilkan pada proses dekripsi. Jika hasilnya cocok , maka algoritme yang telah dibuat telah sesuai dengan algoritme yang di buat oleh Christophe De Canniere dan dapat dikatakan bersifat valid. Nilai *key* dan *IV* diberi nilai *default* masing masing adalah "matahari" dan "bersinar", untuk sample *plaintext* yang akan digunakan adalah dengan *file *.pdf* dengan ukuran 31 Kb yang merupakan ukuran tengah-tengah dari kekuatan sistem yakni maksimal 40 Kb.

6.3.3 Hasil dan Pembahasan

Pada hasil pengujian ini *plaintext input, ciphertext* dan *plaintext output* yang di tampilkan adalah dalam bentuk *hexadecimal*.

Key = "matahari"

IV = "bersinar"

Plaint text *input = file *.pdf* dengan ukuran 31 Kb.

Tabel 6.2 Pengujian Enkripsi

Plaintext input	*Ciphertext*	Keterangan
0x14185918481CD8585D081A5B9A4818985B9E585AC81CD95AD85B1	0x8AE7B2E31125A5779BC5E591C399FE8640ADA012E83AB8CB437C12	Valid

Tabel 6.3 Pengujian Dekripsi

Plaintext input	*Plaintext output*	Keterangan
0x14185918481CD8585D081A5B9A4818985B9E585AC81CD95AD85B1	0x14185918481CD8585D081A5B9A4818985B9E585AC81CD95AD85B1	Valid

Dari pengujian diatas terdapat 2 kesimpulan yakni yang pertama adalah pada Tabel 6.2 menunjukkan perubahan dari *hexadecimal input* awal menjadi *hexadecimal* yang berbeda dan jikalau dirubah menjadi kode *ASCII* akan menampilkan karakter yang tidak terbaca, sehingga pengujian proses enkripsi ini bisa disimpulkan valid. Pada tabel 6.3 menunjukkan adanya persamaan antara *plaintext input* dan *plaintext output* sehingga dapat disimpulkan bahwa proses dekripsi bisa dikatakan valid. Sehingga dari kedua kesimpulan tersebut menunjukkan bahwa algoritme yang telah dibuat sudah sesuai dengan ketentuan yang ada di dalam paper Trivium Stream Cipher.

6.4 Pengujian *Availability*

6.4.1 Tujuan pengujian

Pengujian ini bertujuan untuk mengetahui apakah algoritme Shamir's Secret Sharing yang digunakan sudah berjalan dengan baik dan sesuai dengan pembuat algoritme Shamir's Secret Sharing. Pengujian *split* bisa di anggap berhasil jika data yang di pecah dapat terbagi 3 hasil *share* yang berbeda dan sesuai dengan perhitungan manual yang telah dilakukan oleh penulis . Pengujian *reconstruct* bisa di anggap berhasil jika 2 kombinasi dari hasil *share* yang tersedia dapat mengembalikan data semula.

6.4.2 Prosedur pengujian

Prosedur pengujian yang di lakukan pada pengujian ini adalah dengan cara memproses *split* dan *reconstruct* terhadap nilai *cipher* yang telah dihasilkan pada algoritme Trivium Stream Cipher, tepatnya pada proses enkripsi dengan ukuran *file* 31 Kb dan sudah dirubah menjadi bilangan *decimal*. Nilai *split* dan *reconstruct* yang digunakan adalah 3 untuk *split* dan 2 untuk *reconstruct*.

6.4.3 Hasil dan Pembahasan

Pada sub bab ini akan di tuliskan hasil pembahasan dari pengujian algoritme Shamir's Secret Sharing. Hasil pengujian *split* dapat dilihat pada Tabel 6.4 dan hasil pengujian *reconstruct* dapat dilihat pada Tabel 6.4.

Tabel 6.4 Variasi *Split* Menjadi 3 *Share*

Data
0101000001100001011001000110000100100000011100110110000010110000 1011101000010000001101001011011100110100100100000011000100110000 0101101110011111001011000010110101011001000000111001101100101011010 0110110000101101100011010010010000001101101011001010110010001100 1001011000010010000001111001011000001011011100110011100100000011 0110101100101011011010111010101100100011000010110100001101011010 1000010110111000100000011010110110100101110100011000010……..
Share Ke 1
3008989490243193830140015125818445249526781575684113558270029746 6619840974068036555020470055733357952444617485954560690857283254 0655401337387623371929608236458365799948474596078548009143120085 6548801068981890820754703222017900772468577802616685247547124 3017767709975123032646218728812366710964080538543119028667921394 3676218323232328103058400142109161284897130568781641778929245602 7531135944816448107031792053311276184549256978972974862095016202 5738286386740835512250154399308243051463281485754320575178280 6829066238173817272340425967247841524878497187155380544849501106 0613488215056172986796882584324047919460224924195567028509884

1178249404110079479012776550201861207071789186702903843321015611839890899116543982959987728132557610
Share Ke 2
3319073766582975986018986346408923047908063844416961443634386292576375133927557894385414757633481720236808320951128026452051536715588330759154718896329697835917015483766746754537046728819570120268146490233934707515498912292689260644577436525639585994680199167596379185231236809228937262954093556817723723772727490763234216097315251538287176035394361317982338179347061985596044010823500559384327310470952676798572409942038696373076556449134152501840970084844343640557839851889357881083027670348409418155278441387589315979216621323489530952585692466803005756764583111093724460784673082149698831950755442317258031319832368372073469352110395517230686766525984491953315288721357382459915554450850237993200211163268346492776104334619720
Share Ke 3
3629158042922758141897957566999400846289346113149809328998742838532909293787079233750359459533605488028999155947695362046819819365636527779547100599698572025997450972686033901219292543327131675048282290648961207483965602567477748820577070434593924442236096558421760858159208996584993289237090705581593135642588189387031670011398179795543768069367631023115705053006307553402795565619469982823838172834873561543833543699528135767180138036173288801108111531014603729890664263847891457019727192121386778792728814484516250220260519242719389426756136148634139959723717727176388308081131108126410867018926560586596602414740541177118428823042541630351294054039192433704769370370925578216927265580684864146509523209992710025564480536681830
Keterangan
Valid

Dapat dilihat pada Tabel 6.4 dapat ditarik kesimpulan bahwasanya *ciphertext* yang telah dihasilkan oleh algoritme Trivium Stream Cipher telah berhasil di pecah menjadi 3 bagian (*share*). Nilai *decimal* dari 3 pecahan tersebut telah sesuai dengan perhitungan manual penulis, sehingga pada pengujian *split* ini dapat dikatakan valid.

Tabel 6.5 Kombinasi *Reconstruct* Dari 2 *Share*

No	Kombinasi	Total *Reconstruct*	Status	Keterangan
1	*Share1, Share2*	2	Terpenuhi	Valid
2	*Share1, Share3*	2	Terpenuhi	Valid

3	*Share2, Share3*	2	Terpenuhi	Valid

Dapat dilihat pada Tabel 6.5 dapat ditarik kesimpulan bahwasanya dengan menggunakan kombinasi 2 bagian (*share*), mampu untuk mengembalikan *ciphertext* yang sesuai dengan *ciphertext* yang telah dihasilkan oleh algoritme *trivium stream cipher.* Sehingga pada pengujian *reconstruct* ini dapat dikatakan valid.

6.5 Pengujian Validitas *File*

6.5.1 Tujuan Pengujian

Pengujian ini bertujuan untuk mengetahui *file* yang di hasilkan oleh proses Trivium Stream Cipher dan Shamir's Secret Sharing, dapat sesuai dengan *file* yang asli. Jika telah sesuai maka bisa dikatakan *file* itu valid.

6.5.2 Prosedur Pengujian

Prosedur pengujian yang di lakukan adalah dengan cara mencocokkan isi *file* yang asli dengan isi *file* yang telah di proses melalui algoritme Trivium Stream Cipher dan Shamir's Secret Sharing*. File* yang digunakan adalah *file *.pdf* dengan ukuran 15 Kb, 31 Kb dan 40 Kb. Penguji menggunakan 3 variasi *file* karena disesuaikan dengan kekuatan sistem yakni maksimal 40 Kb, sehingga diambil nilai batas terendah 15 Kb dan nilai tengah 31 Kb.

6.5.3 Hasil dan Pembahasan

Pada sub bab ini akan di tuliskan hasil pembahasan dari pengujian validitas *file *.pdf* yang telah di proses menggunakan algoritme Trivium Stream Cipher dan Shamir's Secret Sharing. Data yang di gunakan untuk memproses *split* dan *reconstruct* adalah data *cipher* yang di dapat saat menjalankan algoritme Trivium Stream Cipher. Hasil pengujian validitas *file* dapat dilihat pada tabel 6.6.

Tabel 6.6 Pengujian Validitas *File*

No	Ukuran *File*	Enkripsi	Dekripsi	*Split*	*Reconstruct*	Hasil *File*
1	15 Kb	Terpenuhi	Terpenuhi	3	2 (*Share1, Share2*)	Valid
2	15 Kb	Terpenuhi	Terpenuhi	3	2 (*Share1, Share3*)	Valid
3	15 Kb	Terpenuhi	Terpenuhi	3	2 (*Share2, Share3*)	Valid
4	31 Kb	Terpenuhi	Terpenuhi	3	2 (*Share1, Share2*)	Valid
5	31 Kb	Terpenuhi	Terpenuhi	3	2 (*Share1, Share3*)	Valid
6	31 Kb	Terpenuhi	Terpenuhi	3	2 (*Share2, Share3*)	Valid
7	40 Kb	Terpenuhi	Terpenuhi	3	2 (*Share1, Share2*)	Valid
8	40 Kb	Terpenuhi	Terpenuhi	3	2 (*Share1, Share3*)	Valid
9	40 Kb	Terpenuhi	Terpenuhi	3	2 (*Share2, Share3*)	Valid

Dari tabel 6.6 dapat ditarik kesimpulan bahwasanya melalui proses enkripsi dan dekripsi pada algoritme Trivium Stream Cipher sampai proses *split* dan *reconstruct* pada algoritme Shamir's Secret Sharing , mampu menghasilkan *file* yang sama dengan *file* asli, walaupun dengan berbagai macam kombinasi *reconstruct*. Sehingga pengujian validitas *file* ini dapat dikatakan valid.

6.6 Pengujian Waktu Algoritme Trivium Stream Cipher

6.6.1 Tujuan Pengujian

Pengujian ini bertujuan untuk mengetahui waktu yang yang di butuhkan algoritme Trivium Stream Cipher untuk menjalankan proses pembentukan *keystream,* enkripsi dan dekripsi.

6.6.2 Prosedur Pengujian

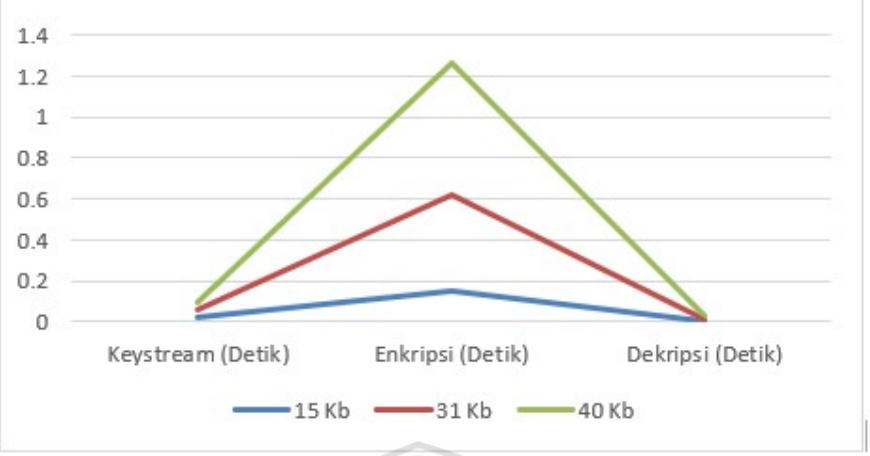
Prosedur pengujian ini akan dilakukan dengan cara diulang sebanyak 30 kali dan diambil rata-ratanya untuk setiap ukuran file yang berbeda dengan *key*, *IV* dan *plain text* yang sama. *Key* yang di gunakan adalah "matahari" dengan maksimal panjang *key* adalah 80 bit atau terdiri dari 10 karakter, dan *IV* yang digunakan adalah "Bersinar" dengan panjang maksimal *IV* adalah 80 bit atau terdiri dari 10 karakter sedangkan *file* yang di gunakan adalah *file *.pdf* dengan ukuran 15 Kb, 31 Kb dan 40 Kb. Penguji menggunakan 3 variasi *file* karena disesuaikan dengan kekuatan sistem yakni maksimal 40 Kb, sehingga diambil nilai batas terendah 15 Kb dan nilai tengah 31 Kb.

6.6.3 Hasil dan Pembahasan

Berikut adalah hasil pengujian yang telah didapatkan dengan mengambil rata-rata dari 30 kali pengujian dari setiap ukuran file **.pdf* yang berbeda.

Tabel 6.7 Hasil Waktu Pengujian Algoritme *Trivium Stream Cipher*

No	Ukuran File	*Keystream (Detik)*	Enkripsi (Detik)	Dekripsi (Detik)
1	15 Kb	0,021	0,153	0,002
2	31 Kb	0,034	0,463	0,011
3	40 Kb	0,042	0,648	0,020



Gambar 6.2 Grafik Waktu Eksekusi Proses Pembentukan *Keystream*, Enkripsi dan Dekripsi

Dari Tabel 6.7 dan Gambar 6.2 menjelaskan tentang waktu eksekusi yang dibutuhkan dalam proses yang terdapat pada algoritme Trivium Stream Cipher, seperti proses pembentukan *keystsream,* proses enkripsi, dan proses dekripsi. Data yang diperoleh adalah untuk *file* ukuran 15 Kb dibutuhkan waktu eksekusi *keystream* adalah 0,021, enkripsi adalah 0,153, dan dekripsi adalah 0,002. Untuk *file* ukuran 31 Kb dibutuhkan waktu eksekusi *keystream* adalah 0,034, enkripsi adalah 0,463, dan deskripsi adalah 0,011. Untuk *file* ukuran 40 Kb dibutuhkan waktu eksekusi *keystream* adalah 0,042, enkripsi adalah 0,648, dan deskripsi adalah 0,020. Shingga dapat disimpulkan bahwa semakin besar *file* yang akan diproses, maka akan semakin banyak waktu yang akan dibutuhkan dalam pemrosesan *keystream,* enkripsi dan dekripsi.

6.7 Pengujian Waktu Algoritme Shamir's Secret Sharing

6.7.1 Tujuan Pengujian

Tujuan pengujian ini adalah untuk mengetahui waktu tempuh proses proses yang ada didalam algoritme Shamir's Secret Sharing. Seperti proses *split* dan *reconstruct.*

6.7.2 Prosedur Pengujian

Prosedur pengujian yang dilakukan adalah dengan cara 3 variasi ukuran *file* *.*pdf* yakni 15 Kb, 31 Kb dan 40 Kb. Jumlah *Split* yang digunakan adalah 3 dan jumlah *reconstruct* yang digunakan adalah 2. Penguji menggunakan 3 variasi *file* karena disesuaikan dengan kekuatan sistem yakni maksimal 40 Kb, sehingga diambil nilai batas terendah 15 Kb dan nilai tengah 31 Kb.

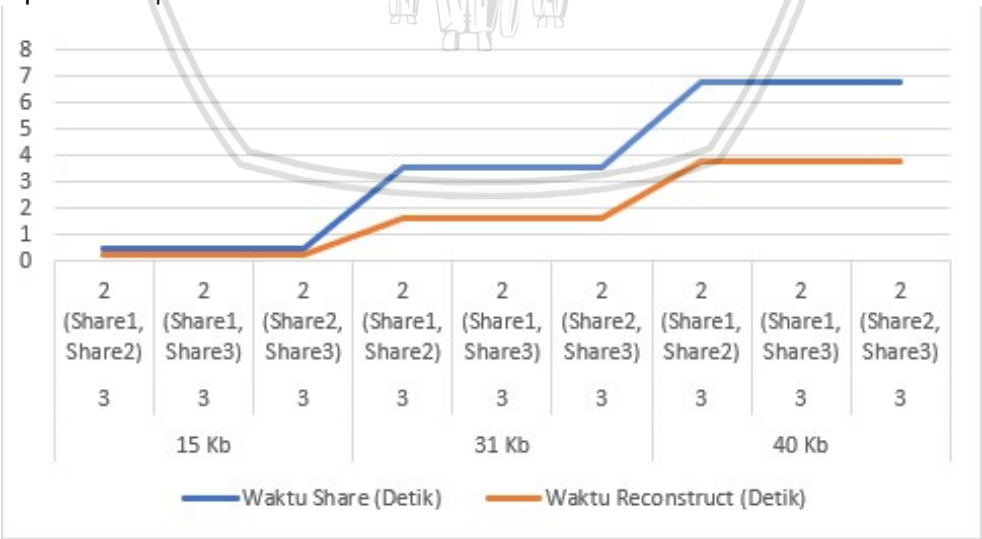
6.7.3 Hasil dan Pembahasan

Berikut merupakan hasil pengujian dari algoritme Shamir's secret sharing.

Tabel 6.8 Hasil Waktu Eksekusi Pengujian Algoritme Shamir's Secret Sharing

No	Ukuran *File*	*Share*	*Reconstruct*	Waktu *Share* (Detik)	Waktu *Reconstruct* (Detik)
1	15 Kb	3	2 (*Share1, Share2*)	0,482	0,210
		3	2 (*Share1, Share3*)	0,483	0,212
		3	2 (*Share2, Share3*)	0,482	0,211
2	31 Kb	3	2 (*Share1, Share2*)	3,563	1,642
		3	2 (*Share1, Share3*)	3,565	1,643
		3	2 (*Share2, Share3*)	3,564	1,642
3	40 Kb	3	2 (*Share1, Share2*)	6,785	3,812
		3	2 (*Share1, Share3*)	6,786	3,813
		3	2 (*Share2, Share3*)	6,785	3,813

Dari Tabel 6.8 menjelaskan waktu tempuh yang dibutuhkan oleh algoritme Shamir's Secret Sharing untuk melakukan proses *share* dan *reconstruct*. Dapat disimpulkan bahwa waktu eksekusi untuk proses *share* dan *reconstruct* pada setiap ukuran file yakni 15 Kb, 31 Kb, dan 40 Kb dengan menggunakan kombinasi *reconstruct* yang berbeda seperti kombinasi *share1* dengan *share2, share 1* dengan *share 3,* dan *share 2* dengan *share 3* adalah tidak adanya perbedaan yang signifikan. Namun jikalau dibandingkan secara waktu keseluruhan, maka semakin besar ukuran *file* makan akan semakin banyak waktu yang dibutuhkan untuk proses *split* dan *reconstruct.* Grafik waktu eksekusi proses *split* dan *reconstruct* dapat dilihat pada Gambar 6.3.



Gambar 6.3 Grafik Waktu Eksekusi Proses *Split* dan *Reconstruct*

6.8 Pengujian Waktu Eksekusi Keseluruhan Sistem

6.8.1 Tujuan Pengujian

Pengujian ini bertujuan untuk mengetahui waktu yang dibutuhkan sistem untuk memproses *file* dengan algoritme Trivium Stream Cipher dan Shamir's Secret Sharing mulai dari proses enkripsi, dekripsi, *split* dan *reconstruct*.

6.8.2 Prosedur Pengujian

Prosedur Pengujian ini akan dilakukan sebanyak 3 variasi dari 2 bagian (*share*) seperti kombinasi *share1* dengan *share2, share 1* dengan *share 3,* dan *share 2* dengan *share 3* pada algoritme Shamir' Secret Sharing . Sasaran pengujian ini adalah pada proses membaca *byte file*,menulis *byte file*, enkripsi, dekripsi, *split, reconstruct*.

6.8.3 Hasil dan Pembahasan

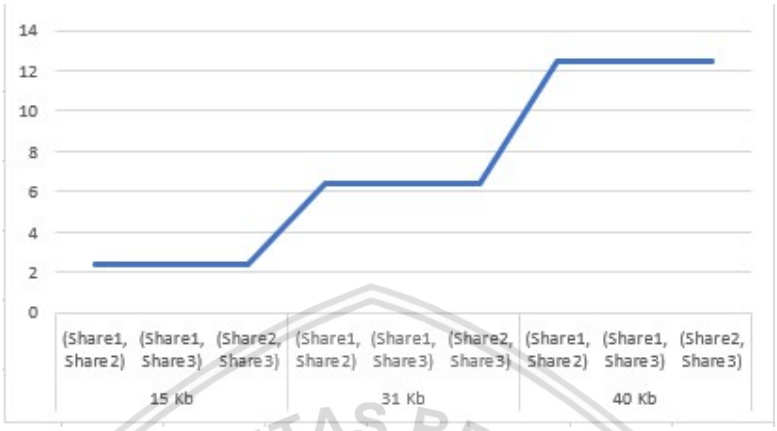
Pengujian terakhir yang digunakan adalah dengan cara menggabungkan seluruh waktu yang dibutuhkan sistem (Proses membaca *file*, enkripsi *file*, *split file*, *reconstruct file* , menulis ulang *file*) dengan pembagian kelompok variasi pada algoritme Shamir's Secret Sharing kemudian menganalisisnya. Untuk hasil pengujian dapat dilihat pada Tabel 6.9.

Tabel 6.9 Hasil Pengujian Waktu Tempuh Sistem Enkripsi dan S*ecret Sharing*

No	Ukuran *File*	Enkripsi	Dekripsi	Split	Reconstruct	Waktu (Detik)
1	15 Kb	Valid	Valid	3	2 (*Share1, Share2*)	2,367
		Valid	Valid	3	2 (*Share1, Share3*)	2,359
		Valid	Valid	3	2 (*Share2, Share3*)	2,365
2	31 Kb	Valid	Valid	3	2 (*Share1, Share2*)	6,386
		Valid	Valid	3	2 (*Share1, Share3*)	6,379
		Valid	Valid	3	2 (*Share2, Share3*)	6,387
3	40 Kb	Valid	Valid	3	2 (*Share1, Share2*)	12,456
		Valid	Valid	3	2 (*Share1, Share3*)	12,459
		Valid	Valid	3	2 (*Share2, Share3*)	12,453

Dapat dilihat pada Tabel 6.9 menunjukkan waktu tempuh yang dihasilkan oleh sistem enkripsi, secret sharing, membaca *byte file* dan menulis *byte file* yang diambil dari setiap kombinasi 2 share. Dari hasil itu dapat disimpulkan bahwa untuk setiap ukuran *file* meliputi 15 Kb, 31 Kb, dan 40 Kb telah berhasil diterapkan algoritme Trivium Stream Cipher dan algoritme Shamir's Secret Sharing. Waktu eksekusi yang dibutuhkan untuk *file* ukuran 15 Kb adalah 2,367 detik, 2,359 detik,

dan 2,365 detik. Waktu eksekusi yang dibutuhkan untuk *file* ukuran 31 Kb adalah 6,386 detik, 6,379 detik, dan 6,387 detik. Waktu eksekusi yang dibutuhkan untuk *file* ukuran 40 Kb adalah 12,456 detik, 12,459 detik, dan 12,453 detik. Grafik hasil waktu eksekusi kedua algoritme dapat dilihat pada Gambar 6.4.



Gambar 6.4 Grafik Hasil Waktu Eksekusi Sistem Enkripsi dan *Secret Sharing*

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan hasil penelitian yang telah dilakukan, dapat disimpulkan beberapa hal mengenai algoritme Trivium Stream Cipher dan Shamir's Secret Sharing.

1. Algoritme Trivium Stream Cipher dapat memberikan aspek *confidentiality*, hal ini dapat dibuktikan dengan pengujian enkripsi dan dekripsi yang telah dilakukan dan bisa dikatakan valid, algoritme Trivium Stream Cipher pada proses enkripsi akan menghasilkan *output* berupa *ciphertext* berupa karakter tidak jelas yang sulit dipahami dan apabila ingin membaca *cipher text* tersebut diharuskan untuk melalui proses dekripsi terlebih dahulu. Pengujian keystream, *test vector* yang dihasilkan pada algoritme ini bisa dikatakan valid.
2. Algoritme Shamir's Secret Sharing dapat memberikan aspek *availability*, hal ini dapat dibuktikan dengan pengujian *share* dan *reconstruct* yang dapat dikatakan valid pada pengujian sebelumnya. Saat suatu *file* dipecah menjadi beberapa bagian, kemudian dapat dikembalikan ke *file* semula hanya dengan jumlah bagian yang lebih sedikit dari jumlah total bagian saat di pecah. Hal ini mengantisipasi saat adanya *file* utama atau bagian *file* yang hilang.
3. Pada algoritme Trivium Stream Cipher menunjukkan bahwa semakin besar ukuran *file *.pdf* yang diuji, maka akan membutuhkan waktu yang lebih besar untuk proses *keystream,* enkripsi dan deskripsi. Pada algoritme Shamir's Secret Sharing menunjukkan variasi kombinasi share pada ukuran *file *.pdf* yang sama, tidak mempengaruhi waktu yang signifikan pada pemrosesan *split* dan *reconstruct.*
4. Hasil akhir waktu eksekusi keseluruhan sistem dengan variasi kombinasi share adalah untuk *file* ukuran 15 Kb adalah 2,367 detik, 2,359 detik, dan 2,365 detik. Waktu eksekusi yang dibutuhkan untuk *file* ukuran 31 Kb adalah 6,386 detik, 6,379 detik, dan 6,387 detik. Waktu eksekusi yang dibutuhkan untuk *file* ukuran 40 Kb adalah 12,456 detik, 12,459 detik, dan 12,453 detik.

7.2 Saran

Saran yang dapat penulis berikan untuk penelitian selanjutnya yaitu dengan menambahkan aspek aspek keamanan lainya untuk diterapkan pada sistem, seperti aspek *integrity, non-repudation, authority* dan *acces control*. Dan juga perlu dilakukan penelitian lebih lanjut dengan menggabungkan algoritme kriptografi yang lainya untuk menambah keamanan. Diharapkan penelitian ini dapat digunakan sebagai rujukan untuk pengembangan selanjutnya.

DAFTAR PUSTAKA

Amin, M. M., 2016. Implementasi Kriptografi Klasik Pada File Text. *Jurnal Pseudocode, Volume III Nomor 2.*

Ariyus, D., 2008. Pengantar Ilmu Kriptografi Teori, Analisis dan Implementasi. Yogyakarta: Andi Offset.

Christophe De Canni`ere, B. P., 2007. Trivium Specifications.

Dimas Aulia Trianggana, H. L. S., 2015. Analisis Perbandingan Kinerja Algoritme Blowfish dan Algoritme Twofish Pada Proses Enkripsi dan Dekripsi. Volume 2 Nomor 1.

Hair, J. B. W. B. B. J. &. A. R. E., 2010. *Multivariate Data Analysis.* 7 ed. Upper Saddle River: Prentice Hall.

Hamdani, 2012. Penerapan Metode Vigenere Pada Kriptografi Klasik Untuk Pesan Rahasia.

Imad Fakhri Al Shaikhli, A. M. Z. H. M.-S. K. P., 2012. Protection of Integrity and Ownership of PDF Documents using Invisible Signature.

Kurniawan, Y., 2004. *Kemanan Internet dan Jaringan Telekomunikasi.* Bandung: Informatika Bandung.

Satria Prayudi, R. R., 2015. Analisis Keamanan Pada Kombinasi Protokol Secret Sharing dan Three-Pass.

Schildt, H., 2007. *Java : The Complete Reference.* New York: McGraw-Hill Education.

Shamir, A., 1979. How to Share a Secret.

Tarbudi, 2013. Membangun Aplikasi Keamanan Transmisi Data Multimedia Menggunakan Kriptografi Algoritme Data Encryption Standard (DES).